KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Applied Deep Learning in Security and Privacy

Vera Rimmer

Supervisors:
Prof. dr. ir. W. Joosen
Dr. ir. D. Preuveneers

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

December 2022

Applied Deep Learning in Security and Privacy

Vera RIMMER

Examination committee:
Prof. dr. ir. D. Vandermeulen, chair
Prof. dr. ir. W. Joosen, supervisor
Dr. ir. D. Preuveneers, supervisor
Dr. ir. L. Desmet
Prof. dr. J. Davis
Prof. dr. U. Meyer
(RWTH Aachen University)
Prof. dr. L. Cavallaro
(University College London)
Prof. dr. K. Rieck
(Technische Universität Braunschweig)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

December 2022

© 2022 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Vera Rimmer, Celestijnenlaan 200A box 2402, B-3001 Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

Preface

This dissertation is a long-awaited culmination of six years of research work. I selected a challenging topic that fascinates and humbles me greatly to this day. Pursuing a PhD in your twenties, a time of self-discovery and adventure, must profoundly impact one's still forming personality – though I will never know for sure. What I do know is that the lessons, connections, and memories this process has yielded are everlasting. Mind you, it was not all smooth sailing. The last years have been darkened by wars and the global health and climate crises. These times are testing our spirit, determination, and faith in the future. Luckily, this highly demanding academic and personal journey has been supported by many, and I want to thank everyone for the roles they have played in my arrival at the destination.

First and foremost, I extend my heartfelt gratitude to my promotors, Wouter Joosen and Davy Preuveneers. They have trusted me with this task and contributed generously to my growth. Wouter, you are a true global thinker, with an instinct for impact and a heart for science. I am sure that the pearls of wisdom you have shared over the years will help me navigate my future life and career. Thank you for helping me put my strengths and interests to use and empowering me to contribute to the lab in various ways. Your approach to understanding and valuing every individual working with you is priceless. Davy, you have believed in me from day one, and your invariable confidence in my abilities helped me, too, to believe in myself. Your inquiring mind and all-encompassing knowledge make for stimulating conversations and exciting ideas, and I have learned a great deal from you. I sincerely appreciate you for supporting my independence while at the same time always being there when I needed you.

I also express profound appreciation to the other members of my doctoral committee: Lieven Desmet, Jesse Davis, Ulrike Meyer, Lorenzo Cavallaro, and Konrad Rieck. It is an honor and a pleasure to have my work and ideas assessed by such insightful and enthusiastic scientists as you. Thank you for the time,

attention, and unique expertise you have put into reviewing my manuscript and into our engaging discussions.

The list of my mentors would be incomplete without Marc Juarez, who was my thesis advisor in my AI master's. Marc, you were the person to welcome me to the world of academia, and in that I believe myself to be very fortunate. Working alongside you has taught me how computer science can directly impact society and human rights. Your sense of purpose, scientific curiosity, and integrity have inspired me greatly, and I intend to hold on to these values in my professional journey. I am also happy to call you my friend.

I am grateful to all my other collaborators for their direct contribution to my PhD trajectory and this dissertation, and I would like to extend some special thanks. Tom, it has been genuinely great to work with you. Your impressive skills, humor, and optimistic attitude have managed to turn paper submissions into pleasant memories. Ilias, I am lucky to have not just a bright and talented coworker in you but also a kind and cheerful friend. Abel, Gints, Ali – thank you all for becoming an invaluable part of the team and for trusting me with the start of your research careers. In the last two years, whenever it has been tough to make sense of things, working with you and helping you to the extent of my abilities has given my working days meaning. Katharina, the project under your supervision has been a pinnacle of teamwork during my PhD: you are a great leader and the loveliest human being, and I hope we will have fruitful (and fun!) collaborations in the future. Roberto, I am very grateful to have met you and been hosted by you and your team at FBK in beautiful Trento. I enjoyed and learned a lot from our long chats and project discussions. And, of course, Tim, my PhD buddy – thank you for your kind and reliable companionship and our uplifting conversations about work projects and beyond.

Being part of DistriNet means belonging to a vibrant, authentic, and friendly environment. The prominent people responsible are our business office. I have mostly worked with Katrien and Annick, and I am very grateful for your gracious help and all our conversations on countless topics. In addition to those already mentioned, I want to thank these people for the joyful time during my PhD: Giuseppe, Fritz, Emad, Thomas, Jan, Job, Neline, Stelios, Gertjan, Andreas, Victor, Yana, Jan Tobias, Leslie, Kim, Vik, Marton, and others. Thank you all!

While a job is but one part of one's identity, my doctorate has been strongly interconnected with other aspects of life. My friendships in and out of Belgium have played an enormous role in this achievement. I want to thank all my friends for being there for me, for taking a sincere interest in my work, caring for my well-being, celebrating my successes, and bearing my sorrows with me. Although I cannot possibly put everyone here by name, you certainly know who you are and that I value you a lot. From my side, I can promise you that you

will never hear this phrase from me anymore: "Sorry, I can't! I have a deadline." Being wonderful friends that you are, you'll hopefully forgive me the moment I break this promise, which will probably come rather soon.

I can hardly put in words the depth of gratitude, love, and appreciation for my family, who have been there every step of the way. But I shall try. My parents are, to the best of our knowledge, the most loving parents walking this Earth. They have been encouraging me to get educated and to stand on my own feet for as long as I can remember, never through pressure but only through inspiration and support. When I moved abroad, which was painful for all of us, they had my interests and future at heart. I will be forever grateful to them for accepting my decision. Thankfully, Belgian chocolate and beer have turned out quite effective at softening the blow. I want to thank my dear brother, who is also my close friend and supporter (that is, ever since we stopped sharing a room ten years ago), and our family cat Milka, who would often stay up all night with me during exams back in the university. This PhD would have never been finished without my family's never-ceasing words of encouragement and optimism. This also includes my extended family: my godmother, grandmother, and my partner's family. I love you all — я очень вас всех люблю.

Finally, I want to thank my tiny nuclear family: my like-minded soul and partner, Jo, and our cutest cat Snezhok. They have been looking after me and keeping me sane during this last sprint. Jo, there seems to be no limit to your appreciation and support for me, and I hope you know that this result is as much yours as it is mine. Thank you for being on my team. I often feel that we are one person facing this world together, able to do just about anything we set our minds to. Cannot wait to find out what the next chapter brings.

The game is afoot!

— Vera Rimmer
November 2022, Leuven

Abstract

Today's digital society is threatened by evermore sophisticated cyber attacks. One of the drivers behind this phenomenon is the advances in Artificial Intelligence (AI), which empower attackers to perform automated analysis of victim systems. Thankfully, AI can also be leveraged to upgrade defensive systems to match the unprecedented speed and scale of attacks. This duality of use motivates comprehensive research into and scientific understanding of both offensive and defensive applications of AI in cybersecurity.

In this dissertation, we investigate deep learning as a compelling data-driven AI technique in the realm of network security and privacy. These domains deal with network traffic characterized by overwhelming volumes, dynamic nature, and low interpretability by human analysts. Hence, we focus on applying deep learning techniques to automatically derive high-level representations from network traffic for offensive and defensive approaches. The overarching objective is then to assess general applicability, advantages, and possible drawbacks of deep neural networks applied to security and privacy.

In the first part of this dissertation, we research side-channel traffic analysis attacks on Tor, a popular privacy-enhancing technology on the web. As a first contribution, we develop deep learning techniques to deanonymize encrypted Tor traffic automatically. Our results, for the first time, obviated the need for manually engineering traffic features, as required by prior website fingerprinting attacks. This work has proven to be a game changer in this area, leading to further active development of deep learning-based website fingerprinting attacks and more robust defenses.

As a second contribution, we propose a more realistic data collection setup to diminish timing bias when modeling so-called end-to-end traffic confirmation attacks on Tor in a lab environment. These results show that deep neural networks, now the established state-of-the-art approach for traffic analysis on Tor, silently memorize artifacts in data, which may cause a drastic overestimation of

attack performance. We develop a general scientific methodology for replicating and comparing attacks while considering the non-determinism and variability of deep neural networks.

In the second part of this dissertation, we investigate the promises and pitfalls of deep learning for defensive applications in network intrusion detection. First, we comprehensively overview the foundations, recent advances, and open challenges for data-driven network intrusion detection. Next, we present an empirical analysis of the applicability and reliability of deep neural networks as a stand-alone mechanism for analyzing raw traffic and detecting malicious or suspicious behaviors. These results highlight some of the inherent critical drawbacks of applying deep neural networks to real-world defensive applications.

In summary, our contributions and resulting insights provide a solid ground for further scientific advances in data-driven security and privacy research while promoting a holistic view of the decision-making capacities of deep learning.

Beknopte samenvatting

Onze digitale samenleving ligt steeds meer onder vuur door gesofisticeerde cyberaanvallen. De vooruitgang in de artificiële intelligentie (AI) draagt sterk bij tot dit gegeven: aanvallers worden bijvoorbeeld in staat gesteld om geautomatiseerde analyses van slachtoffersystemen uit te voeren. Bij het counteren van de ongeziene snelheid en schaal van aanvallen kan men AI-technieken gelukkig ook aanwenden om verdedigingssystemen te versterken en te verbeteren. Dit gegeven motiveert uitgebreid onderzoek naar kennis en inzicht in zowel offensieve als defensieve toepassingen van AI in cyberbeveiliging.

Dit proefschrift onderzoekt deep learning – een bijzonder krachtige, datagestuurde AI-techniek – in het toepassingsgebied van netwerkbeveiliging en privacy. Deze domeinen gaan om met overweldigende hoeveelheden netwerkverkeer, wat resulteert in dynamische data die moeilijk te interpreteren valt door menselijke analisten. We richten ons daarom op het aanwenden van deep learning-technieken om automatisch kenmerken uit netwerkverkeer af te leiden, en dit zowel voor offensieve als defensieve toepassingen .

In het eerste deel van dit proefschrift onderzoeken we side-channel aanvallen door verkeersanalyse op het Tor-netwerk, een populaire privacyverhogende internettechnologie. Als eerste bijdrage ontwikkelen we deep learning technieken om versleuteld Tor-verkeer automatisch te deanonimiseren. In tegenstelling tot voorgaande zogenaamde website fingerprinting-aanvallen, maakten onze resultaten voor het eerst het handmatig achterhalen van onstabiele verkeerskenmerken voor deanonimisatie geheel overbodig. Dit werk vormt een doorbraak en heeft geleid tot een actieve verdere ontwikkeling van deep learning-gebaseerde website fingerprinting-aanvallen en tot sterkere verdedigingen.

Als tweede bijdrage stellen we een meer realistische experimentele opstelling voor om ongewenste vertekening te verminderen bij het modelleren van zogenaamde end-to-end verkeerscorrelatieaanvallen op Tor in een laboratoriumomgeving. Deze resultaten tonen aan dat diepe neurale netwerken, nu de gevestigde state-

of-the-art benadering voor verkeersanalyse op Tor, stilzwijgend kunnen afgaan op irrelevante data-artefacten. In dat geval kan een drastische overschatting van de resulterende aanvalsprestaties optreden. Om deze uitdaging tegemoet te komen, ontwikkelen we een algemene wetenschappelijke methodologie voor het repliceren en vergelijken van aanvallen. Daarbij wordt er rekening gehouden met het niet-determinisme en de inherente variabiliteit van diepe neurale netwerken.

In het tweede deel van dit proefschrift onderzoeken we de beloften en valkuilen voor het gebruik van deep learning bij defensieve toepassingen, met name bij netwerkintrusiedetectie. Eerst geven we een uitgebreid overzicht van de grondslagen, recente vooruitgang en open uitdagingen voor datagestuurde netwerkintrusiedetectie. Vervolgens beschouwen we in een empirische analyse de toepasbaarheid en betrouwbaarheid van diepe neurale netwerken als een op zichzelf staand mechanisme voor het analyseren van ongefilterd netwerkverkeer en voor het detecteren van kwaadaardig of verdacht gedrag. Deze resultaten belichten enkele van de inherente nadelen van het gebruik van diepe neurale netwerken in reële defensieve toepassingen.

De bijdragen en resulterende inzichten van dit proefschrift vormen een solide basis voor verdere wetenschappelijke vooruitgang in datagestuurde digitale beveiliging en privacy, en bevorderen tegelijkertijd een holistische kijk op de besluitvormingscapaciteiten van deep learning.

Contents

Abstract v

Beknopte samenvatting vii

Contents ix

List of Figures xiii

List of Tables xv

List of Acronyms xvii

1 Introduction 1
- 1.1 Expectations of Deep Learning in Security . . . 2
- 1.2 Motivation and Goal of the Thesis . . . 3
- 1.3 Applications . . . 5
  - 1.3.1 Traffic Deanonymization in the Tor Network . . . 5
  - 1.3.2 Network Intrusion Detection . . . 10
- 1.4 Other Contributions . . . 12
- 1.5 Dissertation Outline . . . 16

2 Applied Deep Learning 17
- 2.1 Artificial Neural Networks . . . 17
- 2.2 Deep Learning . . . 22
- 2.3 Understanding the Complexities . . . 23
  - 2.3.1 Evaluation . . . 24
  - 2.3.2 Reproducibility . . . 26
- 2.4 Architectures . . . 26
  - 2.4.1 Stacked Denoising Autoencoder . . . 27
  - 2.4.2 Convolutional Neural Network . . . 29
  - 2.4.3 Temporal Convolutional Neural Network . . . 31

2.5 Conclusion 32

I Traffic Deanonymization 33

3 Automated Website Fingerprinting through Deep Learning 35
3.1 Introduction 37
3.2 Background 39
3.3 Threat Model 42
3.4 Data Collection 43
3.4.1 Data Collection Methodology 43
3.4.2 Datasets 44
3.4.3 Ethical Considerations and Data Access 46
3.5 Evaluation 47
3.5.1 Reevaluation of the State-of-the-Art 47
3.5.2 Deep Learning for Website Fingerprinting 50
3.6 Discussion 63
3.7 Conclusion 64

4 Traffic Correlation Analysis through Deep Learning 65
4.1 Introduction 66
4.2 Technical Background 68
4.2.1 Circuits and Connections 68
4.2.2 Traffic Analysis Attacks 69
4.3 Challenges of Data Collection 69
4.3.1 Measurement Setup and Overview 70
4.3.2 Analysis of RTT Measurements 70
4.4 Data Collection Setup 75
4.4.1 Requirements 76
4.4.2 Setup Overview 77
4.4.3 Data Collection Details 79
4.5 Evaluation Methodology 79
4.5.1 Validity 80
4.5.2 Performance Metrics 81
4.5.3 Attack Replication 82
4.5.4 Explainability Analysis 85
4.6 Experimental Results 86
4.6.1 Experimental Setup 87
4.6.2 Data Input Format 87
4.6.3 Attack Replication: Tuning 88
4.6.4 Attack Replication: Training and Test 91
4.7 Discussion 93
4.8 Related Work 96

4.9 Conclusion 96

II Network Intrusion Detection 99

5 Data-Driven Network Intrusion Detection 101
5.1 Introduction 101
5.2 Network Intrusion Detection 103
5.2.1 Network Threats 104
5.2.2 Network Traffic Monitoring 106
5.3 A Data Analysis Approach 108
5.3.1 Machine Learning for NIDS 112
5.4 Challenges and Advances in NIDS Research 115
5.4.1 Original Premise 115
5.4.2 High Error Rates and Performance Estimation 118
5.4.3 Representative Datasets and Ground Truth 119
5.4.4 Concept Drift 121
5.4.5 Real-Time Detection 122
5.4.6 Adversarial Robustness 124
5.5 Conclusion 126

6 Deep Learning for Network Intrusion Detection 129
6.1 Preamble 129
6.2 Introduction 130
6.3 Network Intrusion Detection through Sequential Feature Learning 134
6.3.1 CIC-IDS-2017 Dataset 135
6.3.2 Deep Learning Models 137
6.3.3 Evaluation and Results 138
6.3.4 Limitations and Open Questions 140
6.4 Failures of Neural Networks on Raw Traffic Representations . . 141
6.4.1 Failure Scenarios 141
6.5 Related Work 144
6.6 Methodology 146
6.6.1 CSE-CIC-IDS2018 Dataset 147
6.6.2 Visualization Analysis Framework 149
6.6.3 Automated Exposure of Shortcuts 152
6.7 Preliminary Evaluation 153
6.8 Discussion and Future Work 156
6.8.1 Exposure of Shortcuts 156
6.8.2 Prevention of Shortcuts 156
6.9 Conclusion 159

7 Conclusion **161**
7.1 Summary of Contributions 161
7.2 Concluding Remarks 165

A Additional Materials for Website Fingerprinting **169**
A.1 Model Architectures 169

B Additional Materials for Tor Traffic Correlation **171**
B.1 Original DeepCorr Performance 171
B.2 Tuning and Training Details 171
B.3 Comparison of Attacks 173
B.4 Explainability Graphs 175

C Additional Materials for Deep Learning for NIDS **181**
C.1 Hyperparameter Tuning Details 181

Bibliography **183**

List of Publications **205**

List of Figures

1.1 The research landscape of deep learning and security and privacy 15

2.1 Model of a neuron. 18
2.2 Feedforward neural network, the most standard DL architecture. 20
2.3 Stacked denoising autoencoder. 27
2.4 Convolutional neural network. 30
2.5 LSTM components: a neural network layer and a LSTM block. . 31
2.6 Architectural elements in a TCN. 32

3.1 Website fingerprinting threat model. 42
3.2 Re-evaluation of traditional WF attacks on new data. 48
3.3 Classification accuracy for a growing number of website traces. 48
3.4 Accuracy, loss, and runtime for a growing number of traces. . . 56
3.5 DL vs. CUMUL for a growing size of the closed world. 56
3.6 DL vs. CUMUL resilience to concept drift. 59
3.7 DL vs. CUMUL in the open world setting. 62

4.1 Overview of exit relay and proxy locations. 71
4.2 Distribution of RTT between client and proxy. 73
4.3 Empirical distributions of end-to-end RTTs. 73
4.4 Distribution of Tor's infrastructure. 75
4.5 End-to-end correlation experimental setup. 78
4.6 Hyperparameter tuning of the $\mathcal{TDC}$ model. 89
4.7 Differences between mean activation maps. 94

6.1 Network intrusion detection threat model. 135
6.2 Functionality of the visualization framework. 150

A.1 Stacked denoising autoencoder for website fingerprinting. . . . 169
A.2 Convolutional neural network for website fingerprinting. 170
A.3 Unrolled LSTM network for website fingerprinting. 170

B.1 Validation AP of tuned DeepCorr configurations. 172
B.2 Hyperparameter tuning of the $\mathcal{TDC}$ model. 174
B.3 Differences between mean activation maps. 179

List of Tables

3.1 CUMUL accuracy for a growing closed world. 49
3.2 Time required to find optimal RBF parameter values. 50
3.3 Tuned hyperparameters of the selected DL models. 54
3.4 Accuracy, loss, and runtime for a growing number of traces. . . 56
3.5 Accuracy, loss, and runtime for a closed world. 57
3.6 DL vs. CUMUL in the open world setting. 62

4.1 Number of inter-continental connections within Tor circuits. . . 72
4.2 Mean end-to-end RTTs and timing distribution overlap. 74
4.3 Tuned hyperparameters of the evaluated $\mathcal{TDC}$ models. 87
4.4 E2E attack performance of the tuned $\mathcal{TDC}$ models. 91

6.1 Connection-based representation. 136
6.2 Time window-based representation. 137
6.3 Performance comparison between anomaly detection models. . 139
6.4 Result of filter activation analysis on LUCID. 143

B.1 Hyperparameters of the $\mathcal{DC}$ model on the multi-proxy dataset. 172
B.2 Top performing hyperparameter tuning runs. 173
B.3 Detailed attack results for single-proxy model. 175
B.4 Detailed attack results for multi-proxy model. 176
B.5 Comparison of the impact of uplink vs. downlink timings. . . . 178

C.1 Hyperparameter tuning details for deep learning-based NIDS. . 182

List of Acronyms

1D-CNN	One-Dimensional Convolutional Neural Network
AE	Autoencoder
AI	Artificial Intelligence
ANN	Artificial Neural Network
AP	Average Precision
APT	Advanced Persistent Threat
AS	Autonomous System
AWF	Automated Website Fingerprinting
BiLSTM	Bidirectional Long Short-Term Memory
CNN	Convolutional Neural Network
DAE	Denoising Autoencoder
DDoS	Distributed Denial of Service
DL	Deep Learning
DNN	Deep Neural Network
DoS	Denial of Service
DPI	Deep Packet Inspection
E2E	End to End
FPR	False Positive Rate
GPU	Graphical Processing Unit
HIDS	Host Intrusion Detection System
HT	Hyperparameter Tuning
IDS	Intrusion Detection System
IID	Independent Identical Distribution

IG	Integrated Gradients
IoT	Internet of Things
ISP	Internet Service Provider
KNN	K-Nearest Neighbour
LSTM	Long Short Term Memory
LRP	Layerwise Relevance Propagation
ML	Machine Learning
MSE	Mean Squared Error
MTU	Maximum Transmission Unit
NIDS	Network Intrusion Detection System
NLP	Natural Language Processing
OOD	Out Of Distribution
PCA	Principal Component Analysis
RBF	Radial Basis Function
RF	Random Forest
RNN	Recurrent Neural Network
RTT	Round-Trip Time
SDAE	Stacked Denoising Autoencoder
SGD	Stochastic Gradient Descent
SIEM	Security Information and Event Management
SVM	Support Vector Machine
TBB	Tor Browser Bundle
TCN	Temporal Convolutional Network
TPE	Tree of Parzen Estimators
VA	Visualization Analysis
WF	Website Fingerprinting
XAI	eXplainable Artificial Intelligence

Chapter 1

Introduction

"It is not the answer that enlightens, but the question."

— *Eugene Ionesco*

We live in a society that has learned to utilize and rely on technology to thrive and evolve. Technology has reached a point of no return in its continuous development and impact on our lives. We have come to accept that, given the industry interest and growing resources, no novel opportunity will remain unexplored. The scientific community recognizes the vital value behind exploring and understanding new technologies in order to be aware of both their highest potential and unintended consequences. Thanks to that frame of mind, whenever the next big thing arrives to transform our world for better or for worse, technological advances can be underpinned with scientific understanding to ensure responsible deployment.

In that spirit, when in the mid-2000s the family of Artificial Intelligence (AI) methods called *deep learning* resurfaced after six decades of relative dormancy, scientists and engineers around the world began investigating its uses in various tasks. Deep learning is based on Deep Neural Networks (DNNs), a special and more powerful type of neural networks, which are mathematical functions or models that were originally inspired by the biology of the human brain. Therefore, naturally, the first wildly successful uses of deep learning were demonstrated in complex pattern recognition that had previously been believed to only be possible by humans: primarily targeting domains such as image and speech recognition. These first results have triggered and encouraged supporting applications of deep learning in other major applied AI domains,

ranging from toy problems, such as puzzles and games, all the way to socially and economically impactful and even safety-critical areas, including finance, marketing, entertainment, healthcare, and cybersecurity. Today, few scientific domains remain unaffected by deep learning, and yet, the research journey is far from completed, revealing both answers and new questions.

1.1 Expectations of Deep Learning in Security

The PhD trajectory behind this thesis began in 2016, roughly at the start of the deep learning wave in cybersecurity. At that point in time, Machine Learning (ML) – a traditional set of AI techniques – was considered a mainstream approach to many learning tasks in cybersecurity. ML was already viewed as an indispensable tool for cybersecurity, be that for research purposes or industrial solutions. Its primary goal is efficient and effective automation of security tasks that would allow coping with the ever-growing scale and complexity of the domain. In the context of dynamically evolving malicious activities, ML techniques can allow automatic recognition and analysis of known and discovery of unknown attacks. As such, ML has enabled great progress in such applications as spam detection, traffic identification, vulnerability discovery, malware detection with the discovery of new malware families, detection of software exploits, blocking phishing websites, recognizing and preventing financial fraud, and many more.

Crucially, the holy grail of security analysts is not merely the automation of routine processing of security events and the overwhelming volumes of data they produce but the automation of complex decision-making and subsequent actions that normally only a human specialist could implement. The latter presents countless challenges for ML in any critical area and in cybersecurity in particular. That is mostly because when making impactful decisions, human analysts have to rely on such abilities as the in-depth perception of data, sophisticated knowledge extraction, generalization to wider contexts, complex predictions about the future, and timely adaptation to changes, to name a few. Automating these tasks appeared to be still out of reach, and yet was at the least enticing, if not necessary, to keep up with the scale and evolution of attacks. So when deep learning re-entered the scene, powered with unprecedented computational resources and novel algorithms and architectures, the possibility of transferring its success at solving human tasks from other application areas to cybersecurity (and its neighboring domains such as privacy) could not be overlooked.

In order to understand why the scientific community saw great potential in deep learning for advancing the state-of-the-art in cybersecurity and privacy, we

need to highlight the key difference of deep learning from other ML approaches to intelligent automation (which we refer to as traditional ML). The unique property provided by deep learning is its inherent capability to automatically learn representations from data. In other words, deep learning models do not necessarily require a human in the loop to manually compose meaningful features from the data – they do so automatically when learning from raw data. Moreover, the more representative data is available to the deep learning model, the better the quality of the representation that can be learned in the form of automatically extracted features, and the higher the performance that can be achieved by the model. At the same time, cybersecurity is notoriously characterized by the abundance of heterogeneous raw data of various modalities, which is generated faster than it can be analyzed. Therefore, automatic representation learning (or feature extraction) from raw data could, in theory, enhance security solutions by relaxing the need for domain expertise and human involvement.

Another curious consequence of replacing the combination of manual feature extraction and traditional ML with deep learning is that the learning pipeline becomes less specific to the problem at hand. Since deep learning presents an end-to-end solution that works directly on raw data and outputs the desired result (e. g., a prediction), the same deep learning architectures can be applied across various applications. As long as the raw data has the same format and similar properties to the data from another application, the same deep learning algorithms can be applied to the new data. This opened an opportunity to evaluate the exact models proposed in other domains, e. g., image recognition, to cybersecurity problems, such as spam detection, in an end-to-end fashion. This universality and flexibility of deep learning architectures appeared to be an attractive quality and yielded fast results, often superior to state-of-the-art.

1.2 Motivation and Goal of the Thesis

Despite the loud early successes, the aforementioned theoretical advantages of deep learning could not be taken for granted. Deep learning is still a subset of ML and, as such, is an example of data-driven optimization-based methods, which makes it subject to the "no free lunch" theorem. Specifically, when it comes to applying a given deep learning method to a problem, there is no guaranteed way to correctly predict how well-fitted this particular method is to the problem and its underlying data, even if a similar problem had previously been successfully solved with the very same method. As a result, an application of existing deep learning methods to a novel problem is never guaranteed to have a predictable outcome, despite past experiences. At best, past experience can inform the new application: there is undeniable value in accumulating and

consolidating lessons learned from other applications in terms of best-performing architectures, parameters, metrics, and other aspects.

When the use of deep learning in cybersecurity was scarce, a large number of open questions were posed about its applicability, benefits, and other properties. In my doctoral research, the following general issues were recognized. Firstly, at that point in time, it was still unclear whether deep learning was a generally applicable method for data-driven problems or if it was best aligned with a specific subclass of problems and types of data. More concretely, there was no way of predicting if deep learning would consistently show to be superior – or even feasible – in cybersecurity tasks, especially since the data types in question are in many ways different from images, speech, and other modalities typically found in applied AI research. Specifically, in my research, we have primarily considered network traffic as a data modality of interest and the two associated problems of choice: *traffic deanonymization* and *network intrusion detection*.

Second, the exact practical benefits of representation learning were unclear: it was deemed important to investigate if automated feature extraction through deep learning provided any specific practical benefits that would advance a target application beyond numerical performance improvements. In other words, we aimed to demonstrate whether, in a given security context, the power of deep learning can indeed be attributed to automated feature extraction. One way of assessing that would be to compare the quality of implicit extracted features to the manually-engineered features used in prior approaches. In the context of security and privacy applications, the quality of features would be determined based on their robustness to challenges specific to the domain.

The third open question regarded different types of security research that could leverage the advances in AI. Technology development tends to benefit both attackers and defenders, empowering and even leading the continuous arms race. Therefore, the misuse of deep learning as an offensive technology has also begun to receive generous attention from the research community. In that regard, one of the early objectives of my research was to explore deep learning in both offensive and defensive contexts and, as a result, observe which facets of deep learning shine through, depending on the malicious or benign nature of its usage.

And finally, as the initial research activities unfolded, the fourth major question gradually emerged. Certain basic costs of deep learning had been initially foreseen and expected: i. e., its demands for large amounts of data and computational resources. But it would be wishful thinking to assume that excluding human expert involvement from the feature extraction stage of the learning pipeline would not introduce any additional concerns. In order to conduct a holistic exploration, we set an objective to also recognize the potential

disadvantages of deep learning. Those would encompass, on the one hand, general fundamental weaknesses of the method and how those manifest in security and privacy tasks; and, on the other hand, the weaknesses that could be specific to the cybersecurity domain. In the 2010s, the research community has already become deeply aware of mismatches between the fundamental assumptions of ML as a tool and the practical demands of cybersecurity areas this tool was supposed to propel. Hence, there is value in a more critical assessment of the properties of deep learning in comparison to the requirements of real-world applications.

These primary questions have formed the motivation behind applied deep learning at the heart of my research on security and privacy. For consistency, I will refer to these – not entirely separate – four pillars as (i) applicability, (ii) advantages, (iii) offensive vs. defensive ability, and (iv) inherent disadvantages of deep learning in cybersecurity. The goal of the thesis is, therefore, to assess the power and the impact of applied deep learning in the scope of the selected cybersecurity problems in terms of the aforementioned four properties.

1.3 Applications

In what follows, I briefly describe the two major cybersecurity use cases placed at the core of my research, presented in their respective parts in the thesis. The first use case, in the realm of privacy-enhancing technologies, focuses on the offensive application of deep learning to traffic deanonymization. The second one, in the realm of network security, considers leveraging deep learning as a defensive tool for network intrusion detection. The work and findings that we have made within the scope of these two applications form the main contributions of this dissertation.

1.3.1 Traffic Deanonymization in the Tor Network

The first part of the thesis deals with the analysis of the privacy properties of the Tor network. Tor is a free and open-source anonymous communications technology designed to protect the internet activities of its users against passive traffic observation. Tor provides strong privacy guarantees to its users by routing traffic through a non-predictable series of relays distributed around the globe, applying at least three layers of TLS link encryption. The anonymity property is thus provided by the fact that not a single actor may reveal both the source and the destination of a private connection by having access to its content.

With the efforts of engineers, researchers, and other contributors, Tor is continuously improved and its protection abilities are thoroughly evaluated. This investment is required in order to stay a step ahead of potential adversaries in the real world since malicious eavesdroppers – such as web servers, ISPs, and even nation-state attackers – have partial access to traffic and may attempt to deanonymize it. While Tor uses strong encryption and distributed trust, in theory, it still remains vulnerable to a certain class of traffic analysis attacks, namely side-channel attacks. An adversary may use side-channel information about the protected communication – e. g., its timing and volume metadata – in an attempt to infer sensitive information about it. Investigating this privacy leakage by modeling the attacks in a controlled research environment is crucial for providing adequate defensive mechanisms in Tor.

The Tor community recognizes multiple threat models of various degrees of severity. The two primary attacks considered in research are *website fingerprinting* [89, 104, 154, 170] and *end-to-end traffic correlation* [203, 145, 171]. Website fingerprinting can be realized by a local network eavesdropper who targets a specific user and wishes to determine their sensitive activities on the web. Specifically, the attacker can analyze the traffic between the target user and the entry relay of the Tor network and, with some probability, identify which website the user has visited. The end-to-end correlation attack, on the other hand, assumes a much stronger adversary who has access not only to the traffic entering Tor but also to the traffic leaving Tor to reach the destination web server. The adversary can then attempt to correlate the entry and exit segments by pairing users with their likely destinations.

Automated Website Fingerprinting

Prior to 2017, state-of-the-art website fingerprinting attacks in the literature [86, 154, 223] utilized ML techniques for traffic analysis and demonstrated high fingerprinting abilities in simulated scenarios, threatening privacy guarantees of Tor. However, this prior research has shown that the success of these attacks relies heavily on the particular choice of traffic features used by ML. The features were manually composed and selected by domain experts based on the heuristics of how well they can help to distinguish websites. This is what most of the research effort has been dedicated to composing and selecting the most meaningful salient traffic features for ML. Since the features were designed based on human expert intuition, there was no guarantee that they were optimal for fingerprinting, i. e., capturing the specific characteristics of the web content that could be used to uniquely identify a website or a webpage. Furthermore, manually-selected features can be fragile to changes in traffic, for example, caused by dynamic changes in web content, changes in Tor design, or

deployment of defenses. These issues created the need for an automatic search for new, most optimal features for fingerprinting to allow for a more precise and realistic assessment of privacy risks in Tor.

In our first research project on deanonymization, which began in 2016, the goal is to apply deep learning to conduct a systematic investigation of website fingerprinting attacks through automatic feature extraction. To that end, we explore several families of DNN architectures, which had demonstrated superior performance in other applied AI areas at the time. The underlying objective is to explore the feasibility and possible advantages of deep learning as an offensive technique in traffic analysis research.

Our study [170] is commonly referred to in the literature as Automated Website Fingerprinting (AWF) and forms the basis of Chapter 3 in the thesis. This work makes the following contributions:

- Our study presents the first comprehensive exploration of deep learning algorithms in application to website fingerprinting. We utilize and compare three common architecture families: feedforward networks with an autoencoder, convolutional neural networks, and recurrent neural networks. When correctly tuned and supplied with enough training data, these models are able to learn salient features of monitored websites automatically from minimally processed protected traffic. The attacks show high success rates in different evaluation settings, even with the simplest network configurations.
- We reevaluate state-of-the-art attacks and systematically compare them to deep learning-based website fingerprinting. We show that our attacks reach comparable performance to that of traditional attacks, with slight improvements of up to 2% in attack accuracy while eliminating the need for feature design and selection. Furthermore, deep neural networks are shown to be more robust to concept drift: automatically learned website features appear to be more stable despite the dynamically changing web content over time.
- Until very recently, the collected dataset was the largest website fingerprinting dataset ever gathered. The open world includes 400,000 websites; the closed world consists of 900 websites, with at least 2,500 traffic traces each. We made the generated dataset and the code publicly available, allowing researchers to systematically and reliably evaluate new approaches to website fingerprinting. The dataset has since been widely employed in independent follow-up research on Tor traffic analysis attacks.

Publication Data: V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen. "Automated Website Fingerprinting through Deep Learning". In: *Network and Distributed System Security Symposium*. NDSS '18. The Internet Society, Feb. 2018, pp. 1–15

Methodologies for Data-Driven Traffic Analysis on Tor

The next project on traffic deanonymization presented in this thesis considers end-to-end correlation attacks on Tor [203, 145, 171], which assume powerful adversaries with access to a sufficiently large amount of Tor relays. Traditionally, end-to-end correlation attacks utilized general statistical correlation metrics when making pairwise comparisons between the client- and server-side traffic flows. Concurrently with our project on website fingerprinting through deep learning, the research community has put forward a hypothesis that the general-purpose statistics might not be able to adequately capture the complex nature of the signal in Tor traffic. Deep learning, however, presented an opportunity to learn a correlation metric that would be tailored to the task at hand by being less sensitive to noise and yet receptive to complex, dynamically changing traffic patterns.

The DeepCorr study [145] that came out in 2018 introduced the first deep learning-based end-to-end correlation attack on Tor traffic and demonstrated a very large improvement margin compared to the state of practice. Deep learning not only succeeded at traffic correlation but also required much shorter traffic flows, demonstrated high inference speed, and performed consistently well on varied Tor circuits and web destinations. DeepCorr has confirmed that deep learning techniques hold unprecedented power if exploited by Tor adversaries and thus may escalate the threat of traffic analysis attacks on Tor.

At that point, the motivation behind our second project on deanonymization was formed. Intrigued by the DeepCorr study and the active development of deep learning-based website fingerprinting attacks that followed AWF, we were interested in analyzing the overall impact deep learning had on the field. Generally, it was becoming more and more apparent to the community that deep learning was less of a silver bullet and more of a double-edged sword. Several universal concerns have been voiced in the literature regarding deep learning-based side-channel attacks. On the one hand, there were difficulties in the attack implementation itself: controlling and guiding the tuning of neural network configurations is complex in terms of global optimization and reproducibility. On the other hand, the so-called black-box nature of DNNs does not allow for direct explainability and interpretability of the internal workings of deep learning. This issue hinders the progress in understanding

the leakage the attack exploits and subsequently complicates the development of appropriate countermeasures. Furthermore, regarding more domain-specific concerns, traffic analysis attacks on Tor face fundamental challenges in realistic data collection and attack performance assessment [222, 154]. Despite the performance improvements enabled by deep learning, it is still largely unclear how to realistically assess the severity of traffic analysis attacks in the wild for website fingerprinting and end-to-end correlation.

With these broad challenges in mind, we conducted a follow-up study on DeepCorr that tackles the universal problem of properly understanding deep learning algorithms as they are applied to traffic analysis attacks on Tor. More concretely, we address two key research questions in this realm: (i) the impact of data collection artifacts on simulating deep learning-based end-to-end correlation attacks and (ii) tuning, replicability, and fair comparison of data-driven attacks.

The resulting study [171], covered in Chapter 4, makes the following main contributions:

- We propose a novel data collection setup for end-to-end correlation attacks against Tor. Instead of using a single proxy for simulating these powerful attacks in a research environment, we propose to use multiple proxies strategically distributed around the globe. Our statistical analysis of Tor traffic round-trip timings shows that multiple proxies approximate realistic timing characteristics of user traffic more accurately.

- In order to analyze the impact of the novel setup on estimating the performance of deep learning-based attacks, we devise a general methodology for scientific replication and comparison of data-driven attacks. This methodology applies to cases when the same attack needs to be tuned and fairly compared on different datasets with different properties or when two or more attacks are compared on the same data. The proposed algorithm aims to address the common problem of biases behind evaluating and comparing deep neural network approaches that often occur in practice.

- Our empirical evaluation of a state-of-the-art deep learning-based end-to-end correlation attack (DeepCorr [145]) demonstrates a statistically significant impact of the proposed multi-proxy setup. An average performance decrease of 7.95 % indicates that the prior setup might have led to an overestimated attack performance.

- For this study, we collected a novel Tor traffic dataset, which can be used for both types of attacks: website fingerprinting and end-to-end correlation. The open world consists of 1 million traces, and the closed world consists

of 500 websites. All traces are collected in a live modern Tor network. The dataset's main benefit is that these two attacks can be jointly evaluated on the same data, together with possible countermeasures. This more up-to-date, large dataset will facilitate further research on data-driven traffic analysis on Tor.

Publication Data: V. Rimmer, T. Schnitzler, T. Van Goethem, A. R. Romero, W. Joosen, and K. Kohls. "Trace Oddity: Methodologies for Data-Driven Traffic Analysis on Tor". In: *Proceedings on Privacy Enhancing Technologies* 3, July 2022, pp. 314–335

1.3.2 Network Intrusion Detection

The second part of the thesis explores the qualities and limitations of deep learning from a defensive perspective, specifically in application to network security. Network Intrusion Detection Systems (NIDSs) monitor and analyse traffic in a particular network environment by searching for malicious activities. A NIDS is deployed at strategic points of a network in order to observe all internal and external traffic. As the sophistication and stealth of network-based threats increase, traditional rule-based and probabilistic data-driven ML methods can detect and respond to known and earlier unknown attacks. In the case of the former, a NIDS can recognize a known attack by its template – this is so-called *misuse detection* or *signature-based attack recognition*. Furthermore, advances in AI and ML enable NIDSs to evolve from signature-based systems that rely on rules for pattern matching with known attack signatures to unsupervised anomaly detection approaches that model a normal behavior profile and recognize deviations as potentially unknown attacks. This capability creates an open-world detector and allows us to adapt successfully to the ever-changing threat landscape. ML-driven NIDSs, thus, hold the promise of complementing more human-reliant approaches by realizing both signature-based and open-world detection strategies in a flexible, adaptive, and automated manner.

After the initial success of deep learning at other traffic analysis tasks [120, 228, 9], Deep Neural Networks (DNNs) started being considered for intrusion detection. However, while DNNs can deploy powerful feature extraction, it is unclear how they affect the operational side of network intrusion detection and the trustworthiness of a NIDS. In order to diminish potential recovery costs and prevent a disruption of services, security analysts are interested in verifying that a deployed DNN, which is not explainable by design, behaves as intended. Manually checking and troubleshooting erroneous predictions, be that false alerts or missed intrusions, without direct interpretability of these automated

predictions, is not scalable. In this regard, the black-box nature of DNNs may present a larger issue than a combination of hand-crafted statistical features and shallow ML algorithms, motivating further investigation of this property.

As the first contribution to network intrusion detection area, Chapter 5 presents a wide discussion on the original premise and the current state of ML solutions for NIDS. The content of this discussion is primarily based on our previously published book chapter [169]. As a second contribution, Chapter 6 focuses on exploring whether deep learning can effectively realize network intrusion detection by composing meaningful deep traffic representations and, most importantly, whether the limitations and challenges it presents, such as the added complexity and non-interpretability, can be mitigated in practice.

Based on these goals, we investigate the practical implications of flawed simulations of network traffic in the context of deep learning approaches for NIDS. This research results in several practical contributions: (i) a published improved version [65] of a modern NIDS dataset, where we address multiple traffic simulation flaws; and (ii) a visualization framework aimed at analyzing the causes of non-intended behavior of a deep learning-based NIDS. This framework addresses our objective to analyze the potential limitations of a deep learning-enabled NIDS. We find that deep learning applied to raw traffic succumbs to its inherent vulnerability to *shortcuts* [77], i.e., spurious correlations or overemphasized features in data that create faulty causality links. Similar effects have been observed in other security applications considered in the literature, such as malware detection, vulnerability discovery, and code authorship attribution [230, 13]. As analyzing these effects is highly domain- and data-specific, we limit our investigation to the context of network intrusion detection.

Chapter 6 reports on this exploration. First, the chapter summarizes the outcomes of designing a sequential NIDS pipeline on a corrected benchmark dataset. Next, we expand on the currently ongoing stage of this research, namely, the failure analysis of a deep learning-based NIDS, focusing on exposing its susceptibility to shortcut learning. In summary, the contributions of this research are as follows:

- We overview deep learning algorithms in application to flow-based network intrusion detection in raw traffic representations, as opposed to hand-engineered high-level statistical features. To that end, we explore various DNN architectures, including temporal, recurrent neural networks, and encoder-decoder architectures for an open-world design. Our analysis also includes existing designs in the literature, contributed concurrently to this part of the project.

- As a part of this project, we published a critical study and improved version [65] of a widely popular NIDS dataset (CIC-IDS-2017 [186]) with simulated network behavior, where we address its flow construction validity and labeling logic.
- We provide a critical analysis of performing automated feature extraction from various raw traffic representations, focusing on its limitations and unintended consequences. We show how, in the context of modern, not artifact-free NIDS datasets, neural networks can achieve misleadingly high results while the flawed underlying patterns detected by the model remain undetected. As this happens due to the inherent non-interpretability of the generated predictions, we propose a methodology to systematically expose and analyze artifact-triggered shortcuts learned by models. Our methodology builds upon state-of-the-art explainability algorithms and suggests domain-specific heuristics to guide the search for shortcuts. This contribution is a stepping stone towards establishing standardized and practically accessible error analysis for NIDS research.
- We develop a reusable visualization framework to facilitate a joint exploration of error analysis on NIDSs that work with neural networks and non-interpretable traffic data.

Publication Data: V. Rimmer, A. Nadeem, S. Verwer, D. Preuveneers, and W. Joosen. "Open-World Network Intrusion Detection". In: *Security and Artificial Intelligence*. Springer, Apr. 2022, pp. 254–283

Publication Data: G. Engelen, V. Rimmer, and W. Joosen. "Troubleshooting an Intrusion Detection Dataset: The CICIDS2017 Case Study". In: *2021 IEEE Security and Privacy Workshops (SPW)*. May 2021, pp. 7–12

Ongoing Work: The contributors to the ongoing study on shortcut learning in deep learning-based NIDS are V. Rimmer, R. Doriguzzi-Corin, A. Rodríguez Romero, G. Engelen, I. Tsingenopoulos, D. Preuveneers, D. Siracusa and W. Joosen.

1.4 Other Contributions

The core of this thesis addresses four major questions on the applicability of deep learning in both offensive and defensive applications in traffic deanonymization and network intrusion detection.

At the same time, my PhD trajectory also includes additional research collaborations within the context of security analytics – or security and AI.

Some of these have resulted from my efforts to guide starting PhD candidates in their first research projects. Others directly connect to the team's broader investment in the actively evolving domain of intrusion detection. They all relate to the central theme of applying and strengthening AI in security solutions, such as authentication, malware detection, and adversarial machine learning. For completeness, I briefly summarize these results below.

Troubleshooting an Intrusion Detection Dataset: The CICIDS2017 Case Study. This paper [65], first discussed in the section above, stems from our efforts to critically reflect on and advance our understanding of pitfalls when applying machine learning for network intrusion detection (cf. second part of this dissertation). Particularly, this study presents an in-depth analysis of the modern CIC-IDS-2017 dataset, where we uncovered several flaws in regard to its utility and validity and illustrated the impact they have on ML solutions. Based on our findings, we contribute an improved relabeled CIC-IDS-2017 dataset.

> G. Engelen, V. Rimmer, and W. Joosen. "Troubleshooting an Intrusion Detection Dataset: The CICIDS2017 Case Study". In: *2021 IEEE Security and Privacy Workshops (SPW)*. May 2021, pp. 7–12

Chained Anomaly Detection Models for Federated Learning: An Intrusion Detection Case Study. This paper [159] presents a design that uses blockchain technology to allow federated machine learning models to be audited without relying on centralized training data.

> D. Preuveneers, V. Rimmer, I. Tsingenopoulos, J. Spooren, W. Joosen, and E. Ilie-Zudor. "Chained Anomaly Detection Models for Federated Learning: An Intrusion Detection Case Study". In: *Applied Sciences* 8.12, Dec. 2018, pp. 1–21

Frictionless Authentication Systems: Emerging Trends, Research Challenges and Opportunities. This paper [211] reviews the research area on frictionless authentication systems. In this work, we recognize open research questions and outline priorities and directions for future work.

> T. Van hamme, V. Rimmer, D. Preuveneers, W. Joosen, M. Mustafa, A. Abidin, and E. Argones Rua. "Frictionless Authentication Systems: Emerging Trends, Research Challenges and Opportunities". In: *11th International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2017)*. IARIA, Sept. 2017, pp. 1–5

Fishy Faces: Crafting Adversarial Images to Poison Face Authentication. This paper [75] first explored adversarial poisoning attacks against a realistic

face authentication system, which includes a state-of-the-art deep learning-based feature extractor and a downstream classifier. Particularly, we consider attackers who can partially control some of the user's training data. Our empirical evaluation shows that such attackers can sabotage the effectiveness of the authentication system after inserting a single carefully crafted adversarial image into its training data.

> G. Garofalo, V. Rimmer, T. Van hamme, D. Preuveneers, W. Joosen, et al. "Fishy Faces: Crafting Adversarial Images to Poison Face Authentication". In: *USENIX Workshop on Offensive Technologies*. WOOT '18. Aug. 2018, pp. 1–12

Position Paper: On Advancing Adversarial Malware Generation Using Dynamic Features. This paper [183] critically reviews existing adversarial attacks against ML-based malware detection systems. Particularly, we observe that considerable research has been targeted at developing techniques to evade static analysis approaches, whereas evading dynamic analysis remains largely unexplored. This study aims to draw attention to this gap by identifying a number of practical transformations that can modify behavioral features of malware so as to evade dynamic analysis as well.

> A. Shafiei, V. Rimmer, I. Tsingenopoulos, L. Desmet, and W. Joosen. "Position Paper: On Advancing Adversarial Malware Generation Using Dynamic Features". In: *Proceedings of the 1st Workshop on Robust Malware Analysis*. May 2022, pp. 15–20

Intelligent Malware Defenses. This book chapter [142] reviews the field of ML-based malware defenses. We establish a taxonomy to break this vast research area into the subfields of malware detection, malware analysis, adversarial malware, and author attribution. The resulting overview highlights challenges and research priorities for intelligent malware defenses.

> A. Nadeem, V. Rimmer, W. Joosen, and S. Verwer. "Intelligent Malware Defenses". In: *Security and Artificial Intelligence*. Springer, Apr. 2022, pp. 217–253

Research Landscape. Figure 1.1 depicts the contributions of my doctorate in relation to the general research landscape of deep learning and security and privacy. This includes the primary activities on traffic deanonymization and network intrusion detection, and other contributions. The horizontal dimension in the figure differentiates offensive and defensive research or attacks and defenses. The vertical dimension contrasts deep learning applied to enhance security and privacy (the top row) to evaluating and robustifying deep learning against attacks (the bottom row). The body of this dissertation is formed from the contributions in the top row.

	Offensive applications	Defensive applications
Applied deep learning in security and privacy	Chapter 4 [171] Chapter 3 [170]	Chapter 6 Chapter 5 [169] [141] [65] [159] [211]
Security and privacy properties of deep learning	[183] [75]	—

Figure 1.1: Contributions placed within the general research landscape.

Academic Service. Apart from conducting research, I have also contributed to the domain through academic service. Substantial effort has been put into reviewing research articles in the vast areas of AI/ML, security and privacy. These reviewing contributions have been twice recognized with an award. Here I list such primary activities, including the type of service and the corresponding academic journals and events:

2019 IEEE European Symposium on Security and Privacy (EuroS&P) – external reviewer.

2020 EuroS&P – PC reviewer, session chair.

New Security Paradigms Workshop – PC reviewer.

Workshop on Privacy in the Electronic Society – PC reviewer.

IEEE Transactions on Dependable and Secure Computing – reviewer.

2021 EuroS&P – PC reviewer, session chair.

↪ ★ *Distinguished reviewer award 2021.*

Computer Networks – reviewer.

2022 EuroS&P – PC reviewer, posters co-chair, session chair.

↪ ★ *Distinguished reviewer award 2022.*

ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec) – PC reviewer.

Privacy Enhancing Technologies Symposium (PETS) – external reviewer.

2023 PETS – PC reviewer.

WiSec – PC reviewer.

EuroS&P – mentoring chair.

Workshop on Secure and Trustworthy Deep Learning Systems – PC reviewer.

1.5 Dissertation Outline

The remainder of the dissertation is organized as follows. We start by providing background knowledge on deep learning in Chapter 2, including the main principles, architecture families, and known inherent challenges. The next two chapters constitute the first part of the dissertation on the topic of Tor traffic deanonymization. Chapter 3 presents our study on novel website fingerprinting attacks that utilize deep neural networks to perform automated feature extraction from protected traffic. In Chapter 4, we describe our research on another type of traffic deanonymization attack against Tor, namely end-to-end traffic correlation. This chapter presents and analyzes a novel data collection setup designed for a more realistic simulation of deep learning-based attacks. After that, Chapter 5 introduces the dissertation's second part dedicated to network intrusion detection. This chapter overviews the main concepts behind data-driven NIDSs and provides a broad analysis of open challenges and potential research directions. Chapter 6 continues the part on network intrusion detection by researching deep learning as a defensive mechanism and providing a critical analysis of potential drawbacks of representation learning from raw network traffic. Finally, in Chapter 7, we conclude the dissertation by summarizing the key findings and discussing the path forward.

Chapter 2

Applied Deep Learning

In this chapter, we provide the necessary theoretical foundation to deep learning. We start by giving a general introduction to neural networks (the underlying mechanism of deep learning) and main related concepts. Then we introduce deep learning and give important insights in its operation and basic properties, some of which may present challenges when applying deep learning in research contexts and beyond. Finally, we review deep neural network architecture families that are relevant to this dissertation.

Section 2.1 of this chapter is based on my master thesis on deep learning for website fingerprinting [168]. Section 2.4 is based on the original appendix of my publication on automated website fingerprinting [170].

2.1 Artificial Neural Networks

Machine Learning (ML) shows the ability to learn by inferring new knowledge from some previously given knowledge. To achieve that, ML algorithms perform automated data analysis to find hidden patterns and correlations in data. The exact choice of an algorithm for the task largely depends on the application, the type of data and the requested output, and other domain-specific requirements.

Artificial Neural Networks (ANNs) are one such subset of ML techniques which can be used to build a predictive model: generate an output based on its input. As a modeling tool, it is inspired by human neural system. ANN imitates the structure and the learning process of a brain neural network, at the same time

representing a strong mathematical abstraction of reality. It is used to estimate (or approximate) an unknown function that depends on the inputs and outputs.

An ANN can be characterized by a number of parameters.

Architecture. First of all, an ANN is specified by its architecture. The building component of a network is a neuron, depicted in Figure 2.1, modeled as a static non-linear element. Neurons in a network are connected with each other, each connection i having an interconnection weight w_i – a numerical parameter. Neurons are ordered in layers, and the topological relationships between the neurons define the neural network architecture. Figure 2.2 depicts an example of a fully-connected neural network with three layers of neurons.

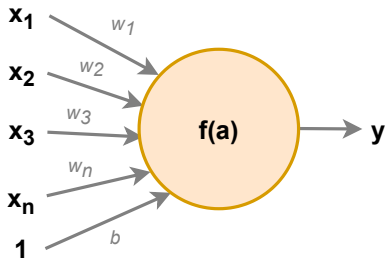


Figure 2.1: Model of a neuron that produces the output $y = f(a)$ for an input $a = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$.

Activation Function. Activation functions are a key component of ANN designs. It is a function that is set on every neuron of the network and defines the type of predictions the model can make. Each neuron computes the sum a of its input values x weighted by the interconnection weights and a bias term or threshold b. Then the neuron applies an activation f to the sum a and computes the output y. The neuron in Figure 2.1 applies an activation function f to a and yields the output y. To get access to a rich hypothesis space, activation functions provide a non-linearity property. There are many possible non-linear functions to choose from, the most common of which are perhaps Rectified Linear Activation (ReLU), Logistic (Sigmoid), and Hyperbolic Tangent (Tanh) functions. To imitate the trigger of a biological neuron on collected information of incoming data that exceeds a certain threshold, the activation function can be chosen with the saturation type of non-linearity. This means having a linear region within input limits, and once the input exceeds the limit, the output becomes constant. This is the case, e.g., for Sigmoid and Tanh activation functions.

Learning Rule. In addition to the architecture and set activation functions, a neural network is characterized by its learning rule (or learning algorithm), which defines its learning behavior. A neural network learns by processing the input data on every neuron (as described before) and changing its interconnection weights according to a specified learning algorithm. The result of learning is the final values of weights which is expected to guarantee correct outputs for new earlier unseen inputs. A trained neural network learns a mathematical and computational mapping from data to decisions, or even the other way around, if its goal is data generation.

Neural networks are known as powerful function approximations. The universal approximation theorem from 1989 [96] states that a feedforward neural network with a single layer is sufficient to approximate any continuous function with an arbitrary precision. While this is a strong promise, in practice there are many caveats to the statement. The theorem gives no guarantees on the number of neurons that are required to achieve this function, and this number could be practically unachievable. Furthermore, it opens a question of how exactly one can find the right weights that could solve the problem. Although the weights exist in theory, obtaining them is not a given. And finally, the theorem does not speak about the final utility of such an approximation. There is no guarantee that the existing approximation will generalize beyond the data used to train the network.

In summary, there is no general algorithm for determining either the network architecture or the values of its interconnection weights for a given non-linear function to estimate. Hence, the process of learning the function by a neural network is generally practice-driven, i.e., the choice of the learning rule for a certain neural network is defined by its architecture, application, input and output data and other influential properties.

Feedforward Neural Network. A standard neural network is a multilayer perceptron, also called a feedforward neural network. A feedforward neural network consists of an input layer, one or more hidden layers and an output layer; each layer with an arbitrary number of neurons. In this "feedforward" type of an architecture the connections between neurons always in one direction, feeding the layer's output as an input to the following layer. An example of a feedforward neural net with one hidden layer is given in Figure 2.2.

Training. Such a network with continuous activation functions is capable of approximating any continuous non-linear function, given non-linear activation functions set on neurons. In order to learn a function, the network has to be trained with data. The training algorithm of neural networks is

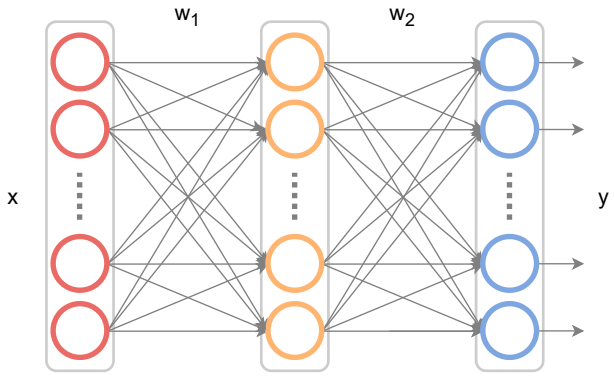


Figure 2.2: Feedforward neural network, the most standard DL architecture.

"backpropagation" [175]. Backpropagation is a supervised learning process based on propagating the prediction error of the output back to the input layer and changing the interconnection weights. The prediction error is computed according to the chosen loss function – the objective function which has to be optimized by the neural network. A traditional choice for feedforward networks is the Mean Squared Error (MSE) computed on the training set of N instances, namely, the summation of squared differences between the predicted output y and the desired output:

$$MSE = \frac{1}{2}\sum_{i=1}^{N}(y_i^{desired} - y_i)^2$$

This loss function is normally used for classification problems, when the output values are probabilities of the instance being classified as the corresponding class.

The goal of the backpropagation process is then to achieve the desired outputs by adjusting the interconnection weights and minimize the chosen loss function. This optimization process is usually performed iteratively, starting with arbitrary initialized weights. In general, backpropagation can be summarized as follows:

1. Forward propagate the input training instances through the neural network and generate the output.
2. Backpropagate the output through the neural network using the target output and compute the differences between the desired and generated output values for all neurons of the network.

3. Randomly sample a small number of data points and compute the corresponding gradient (derivative) of the weights by multiplying the differences by the input values.

4. Adjust the weights by subtracting a certain ratio called learning rate from the weights gradient.

This classic algorithm is called Stochastic Gradient Descent (SGD). The process can be stopped whenever the error measure is sufficiently minimized, its output being the corresponding set of interconnection weights. The most influential parameter of backpropagation is the learning rate used in step 4. The learning rate defines the speed and the quality of learning. The greater its value, the faster the change in interconnection weights, which in turn means that the faster the neural network learns. However, the lower the value of the learning rate, the more accurate the search for an optimal solution becomes. Performing one epoch of training means processing all of the input instances once and adjusting the weights. Backpropagation can be done in batch mode, whereby many propagations occur before finishing the epoch and finally adjusting the weights. The error is then accumulated over the instances in batches of a predefined size so that the average error of the batch is minimized. The size of the batch is then another hyperparameter. The error for every batch is computed in parallel; therefore, increasing the batch size speeds up the learning process. Neural networks can be trained with different versions of optimization algorithms, which normally represent various modifications of SGD.

Generalization and Overfitting. The ability of a model to learn is defined by its generalization capabilities. Generalization refers to a model's ability to describe unseen data. It must fit the training set in such a way that it does not just "memorize" the instances but instead truly captures the underlying relationships in data. This will ensure that the model performs well on other data that follows the same distribution as the training data. The concept of generalization is strongly related to overfitting. The model that does not generalize well to new data usually fails to do so due to overfitting to training data: it learns the noise in the training set, rather than the underlying correlations in data. This can be observed when a model fits the training data perfectly well, but then shows poor performance on newly presented, earlier unseen test data.

Validation. Validation is a part of the training process which is used to select optimal values of the weights and to ensure generalization abilities of the model. After every update of the weights, a network computes also the loss function on the validation set – a separate subset of the dataset which is used only for

validation (i.e., does not take part in backpropagation). Performance on the validation set should be as close as possible to performance on training data, in order to avoid overfitting and to provide a good generalization of the resulting model to new data.

2.2 Deep Learning

Deep Learning (DL) is a subfield of ML that is based on a special type of ANNs, namely Deep Neural Networks (DNNs). A DNN is an ANN that has multiple hidden layers. DNNs have a hierarchical architecture consisting of multiple layers of complex non-linear data transformations. The overall model complexity is proportional to the amount of hidden layers, meaning that deeper models may compute more complex abstractions from the input data.

The deep hierarchical structure and multi-step processing of DNNs distinguishes deep learning from ML, e.g., regular shallow neural networks. Traditional ML on one hand largely relies on prior knowledge of which specific features of the input may help in predicting the right answer. The emphasis in shallow learning is often on feature engineering and selection, specifically on composing the most optimal set of hand-crafted features, guided by domain expertise and heuristics of the target problem. On the other hand, DL obviates manual engineering and selection of features by performing "representation learning", also called "automated feature extraction" or "feature learning". Its common pipeline is as follows: (i) Supplying inputs to a network in their raw numerical form, or with minimal preprocessing, (ii) Passing the inputs through a highly parameterized, multi-layer network that applies non-linear transformations to input values, (iii) Applying a gradient-based optimization algorithm which learns to extract informative representations of the input to approximate the output function. In effect, this process arrives at an accurate solution by learning from minimally processed data with little to no prior knowledge.

While ANNs with one hidden layer have a theoretical guarantee that for any inputs and outputs, the shallow architecture is enough to learn the approximation function; however, in practice this is not a given. As was discussed earlier in the chapter, finding perfect hyperparameters to configure the ANN is largely infeasible. But in addition, intuitively, this guarantee would also rely on finding perfect hand-engineered features to represent the data based on domain expertise and heuristics about the problem. This could explain why adding more layers appears beneficial: deep learning uses multiple hidden layers to chain non-linear transformation on the input data in order to learn the optimal representation of the data automatically. This demands more data, but becomes feasible through

batch processing of data. Moreover, we have more computational power: since 2012, matrix multiplication operation at the heart of neural networks takes advantage of GPUs.

The caveat is that while DL to some noticeable extent relaxes the need for task-specific manual feature generation, it raises the demand for AI-specific knowledge. In summary, it imposes a trade-off between prior knowledge, AI knowledge and practical experience, amount of training data, performance variability, human understanding of the learned model, and many other properties. Therefore, the promise of DNNs to realize complex approximation without domain knowledge often comes with other costs.

DL is favored for the unprecedented performance and efficiency across many computer science domains. A DNN can be used as an independent predictor or even as a powerful feature extractor preceeding a traditional ML algorithm. One security application that is known for utilizing DL for automated feature extraction and other algorithms for prediction is face authentication [181]. DNNs are especially good at directly learning face embeddings by mapping from face images to a compact Euclidean space, where distances correspond to a desired measure of face similarity. These embeddings can be used as feature vectors when recognizing, verifying or clustering identities.

Further in this section, we discuss the nuances and key challenges in designing, evaluating, comparing, and reproducing DNNs. This theoretical background can inform on which considerations need to be adopted for obtaining substantive results in applied DL research, especially in what concerns the race of deep learning techniques that is currently occupying the field.

2.3 Understanding the Complexities

While relaxing the need for domain knowledge through eliminating manual engineering of features, deep learning raises complexity in other ways and is overall not straightforward in deployment. For instance, evaluating deep neural networks and comparing them with each other in a conclusive way can be hard. In the cybersecurity domain, much like in other safety-critical domains, this problem can have far-reaching consequences on the progress in the field. If performance of a defensive or an offensive approach happens to be severely under- or oversestimated, security or privacy of a system in question can be compromised.

Deep learning has entered the cybersecurity domain as a groundbreaking cryptoanalysis technique by advancing end-to-end profiling side-channel attacks

on hardware-backed encryption [125]. Since these first successes, the paradigm has consumed the field: a large portion of research activities in cybersecurity and privacy has been devoted to the development of new, better deep neural networks and their configurations. To evaluate the quality of a new deep learning approach, performance of a trained neural network is compared to the state-of-the-art on earlier unseen data (the test set). In order to assert superiority of the new solution over its predecessors in the task at hand, one needs to demonstrate a convincing performance gap in terms of the chosen meaningful metric(s) for the given application domain. As there is no possibility of formally proving that a new solution is the best data-driven approach for a given application, we as a community make conclusions from limited empirical evidence. These conclusions should generalize to other similar datasets, experiments and model configurations which cannot be evaluated due to data unavailability or computational constraints. Therefore, similarly to other ML approaches, superiority of a new deep learning model or architecture is convincing if there is significant evidence that the empirically observed performance gap will largely generalize to other data in similar contexts. Specifically for deep learning, however, collecting such convincing evidence is overall more challenging, not only in terms of increased computational requirements and volumes of data, but also due to intrinsic characteristics of highly parameterized neural networks.

Evaluation and comparison of data-driven solutions in security and privacy domains has become more complicated with the introduction of DL. Non-convex loss functions, stochastic optimization, random weight initialization and a large space of hyperparameters significantly limit reliability and transparency of DL evaluations. We will gradually unravel these terms and discuss briefly why understanding their role in DNN performance is critical to reliable evaluation of DL-based solutions. This aspect of DL requires close attention in applied data-driven research, since we as a scientific community base our conclusions on limited empirical evidence.

2.3.1 Evaluation

A DNN learns to solve a problem by approximating a mathematical function that maps inputs to desired outputs (e.g., binary files to a malicious or benign label). Training a network means finding good values for its set of *weights* – learnable parameters. Good weights are those that minimize a predefined *loss function* – a function that calculates a cost of erroneous outputs on unseen *validation* data (i.e., the mismatch between predicted outputs and true outputs) and thus intuitively reflects how far the DNN is from solving the problem. DNNs are multi-layered and make use of non-linear transformations, which makes their loss functions *non-convex* – functions that curve up and down,

implying that they have many possible solutions that look optimal within a certain region. Optimal weights cannot be calculated with a formula, but are found through an empirical non-deterministic optimization algorithm, namely SGD. Poorer solutions called *local optima* are easier to find, while the best set of weights called a *global optimum* can be searched for for a very long time.

Randomization. To advance the search and improve chances of finding a global optimum, SGD makes extensive use of randomness. Because nothing is known about the search space at the start, SGD begins with a randomly chosen position by initializing the weights to small random numbers, with various methods and distributions. SGD further randomizes the search by shuffling the data inputs between iterations or *epochs* and by using *dropouts* when certain inputs are randomly ignored. Therefore, running SGD with different random seeds brings different results. SGD *converges* when it stops improving, indicating that the DNN learned an optimal solution. Unfortunately, it is impossible to tell whether the solution is a local or a global optimum. This is why the search process is repeated multiple times, with various random initializations of weights, in order to average various points of convergence, which may differ significantly.

Hyperparameter Tuning. Apart from the chosen method for weights initialization, a DNN's configuration is formed by a multitude of other *hyperparameters* that define its architecture and learning properties. Hyperparameters are preset before training in a controlled manner and can be tuned: an optimal configuration of a DNN will speed up and/or improve convergence, aspiring to reach a global optimum. Hence, finding a proper configuration for a DNN is a meta-optimization problem in itself, which can be attempted either manually or algorithmically. Modern DL algorithms have dozens of hyperparameters to tune, and often only a subset of the most meaningful ones is explored in depth. E.g., the number and the kind of layers and neurons define the structure and the amount of weights in a DNN, which relates to its *learning capacity*. *Learning* hyperparameters define the type, properties and heuristics of the used SGD algorithm, directly influencing the search process. *Regularization* hyperparameters are used to penalize *overfitting* – memorization of training data which makes a model perform worse on unseen data.

As such, randomization during optimization and tuned hyperparameters are the two primary sources of algorithmic non-determinism in DL evaluations, which explains performance variability even in the context of just one DNN trained on the same dataset. As is evident from the discussion, even upon finding a satisfactory point of convergence, there is no promise that searching further will not return a better set of weights for a particular network. In practice, DL

engineers report the metrics achieved within affordable computational time and resources.

2.3.2 Reproducibility

Today, research conclusions drawn from DL attacks commonly rely on success metrics reported on empirically obtained DNN configurations which performed best in a chosen experimental setup on a fixed unseen *test set*. Normally this is meant to highlight a particular DL approach which is superior based on the limited observations. In our discussion above, we expose the fact that DL evaluations are not deterministic and rely on a number of influential factors that are not always explicit or can be taken under control. In summary, these factors include (but are not limited to):

- Chosen evaluation setting and its limitations.
- Choice and size of a dataset, the number of datasets.
- Chosen metrics.
- Selection of tuned hyperparameters and their search spaces, tuning algorithm.
- Possible data leakage and selection bias during tuning
- Computational costs and resources, their consistency across experiments.
- Intrinsic algorithmic randomization during DNN training.

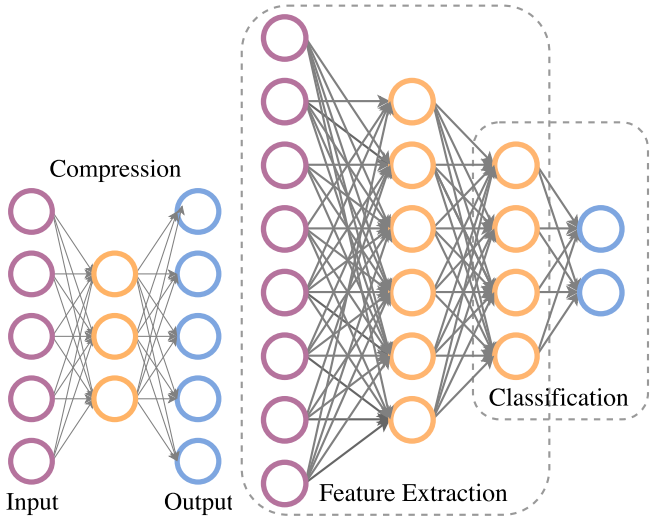
We identify these as sources of performance variability in applied DL solutions, which makes them hard to compare and reproduce. To obtain conclusive results, it is advisable to experiment with various sets of hyperparameters and random seeds. This is especially pronounced when evaluating on new data collected in different conditions. For the same reasons, other applied AI fields, such as Natural Language Processing (NLP), are moving onto *statistical significance* testing of DL evaluations [60], as a replacement for less reliable one-time evaluation scores.

2.4 Architectures

In this section, we review several deep neural network architectures, in addition to the feedforward neural network discussed in Section 2.1. These are the main architectures used in the work presented in this thesis.

2.4.1 Stacked Denoising Autoencoder

An Autoencoder (AE) is a shallow feedforward neural network designed for learning meaningful data representations [218]. It is composed of traditional *dense* layers that work by multiplying matrices of inputs and weights. The basic AE has an input layer, one hidden layer and an output layer, as is shown in Figure 2.3a. The input layer acts as an encoder that transforms data and passes it to the hidden layer $\vec{h} = f(\vec{x})$, and the output layer of the same size acts as a decoder that reconstructs the data back from the hidden layer $\vec{r} = g(\vec{h})$, intending to produce maximally similar values.



(a) Autoencoder **(b)** SDAE from two autoencoders

Figure 2.3: Stacked denoising autoencoder.

The size of the hidden layer plays a crucial role in the AE's working algorithm: it defines the representation of the input used for reconstructing the data. The hidden layer $\vec{h}$ is constrained to have fewer neurons than the input $\vec{x}$. Then such an *undercomplete* AE is forced to compress the input and can only output its approximation rather than the identity. In order to reconstruct the data from a compressed representation with a minimal loss, the network has to prioritize between properties of the data during compression.

AEs will learn certain combinations and transformations of input values that allow to reconstruct the same instance with the highest accuracy. As a result, the hidden layer will contain the most salient features in the input data. The

training is performed by backpropagating reconstruction errors expressed with a loss function that has to be optimized by the network. The loss function $L(\vec{x}, g(f(\vec{x})))$, such as mean squared error, reflects the difference between the input $\vec{x}$ and its reconstruction $g(f(\vec{x}))$, and reaches its minimum value in case of a total similarity between the two. The mean squared error measures the average of squares of the deviations: $L(\vec{x}, g(f(\vec{x}))) = \frac{1}{N}\sum_{i=1}^{N}(g(f(x_i)) - x_i)^2$, where N is the number of neurons of the input (and the output) layer.

Since the undercomplete AE cannot learn a total identity function but only an approximation, its training stops once having minimized the loss function, and thus ensures a good learned representation of data. The AE, as a building block of our future classifier, has to learn representations which reflect statistical properties of the whole data distribution beyond the training examples. This is necessary to achieve a high performance of the model on unseen data, to maximize generalization capability. The AE that during training performs much better than on unseen data, has overfitted to the training data, and thus shows poor generalization capabilities.

To ensure generalization, regularization can be applied by using *dropout*: namely, a randomly chosen fraction of input values is set to 0 at each training iteration. An AE with dropout is called a Denoising Autoencoder (DAE) which is more robust to overfitting [199].

Stacked Denoising Autoencoder is a deep feedforward neural network built from multiple DAEs by stacking them together, in a manner depicted in Figure 2.3b. SDAE stacks the DAEs representation layers: the hidden layer of the first DAE is used as the input layer of the successive DAE, and so forth. Chaining several DAEs enables the model to hierarchically extract data from the input to learn features of different levels of abstraction. Deeper models produce final features of higher abstraction, which are meant to be used for classification on the concluding layer. The classification layer has one neuron for each possible class. Output neurons compute the probability of an input instance belonging to a class. The label of a neuron that produced the maximum probability is assigned to the given instance.

It was discovered by Hinton et al. [91] that in order to achieve a better performing DNN, it has to first be pretrained in an unsupervised fashion, that is without using the knowledge of the ground-truth labels. This strategy is known as the *greedy layer-wise unsupervised pretraining* that initializes the SDAE. This stage is followed by a *supervised fine-tuning* of the whole model that learns to classify the input by backpropagating classification errors. The loss function that expresses the errors is a categorical entropy $E = -\frac{1}{N}\sum_{i}^{N}(p_i log_2 p_i)$, where p_i is a returned probability for the predicted class with N classes in total. A classifier confident of its decisions gives a high probability for each predicted

class, which minimizes the entropy.

2.4.2 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a feedforward network trained with backpropagation that has a different structure from dense layers. CNN is designed for minimal preprocessing [115]. Its main building block is a *convolutional* layer, which performs a linear convolution operation instead of a regular matrix multiplication. The learnable parameters of the convolutional layers are *kernels* (or *filters*) – multidimensional arrays that are convolved with the input data to create *feature maps*, as depicted in Figure 2.4. A kernel is applied spatially to small regions of the input, thus enabling sparse connectivity and reducing parameter learning in comparison to fully-connected layers. Kernels aim to extract specific patterns from an underlying feature set. The convolution function is followed by a non-linear activation, typically a *rectifier* [144]. The rectified feature maps are stacked together along the depth dimension to produce the output.

The next operation of the CNN is typically a *pooling* layer that performs a subsampling operation by replacing the output of the convolution layer with a summary statistics of the nearby outputs. For instance, a max pooling layer reports the maximum outputs within regions of the feature maps. Pooling not only reduces dimensionality, but also helps to make the representation invariant to minor changes. Specifically, such subsampling may allow to find prominent patterns within the input, despite their slight shifts in location and the surrounding information.

CNNs are widely used in computer vision tasks, where they operate on images. In the case of images, the CNN's input space is 2-dimensional, while in temporal domains (like speech, auido or network traffic), it is 1-dimensional. For a sequence of length T and number of features N, the 1D convolutional kernel is a $N \times S$ window, where S is the size of its receptive field. The kernel slides over the sequence from start to finish, similar to the well-known convolution operation in signal processing. This process is repeated for a number K of independent kernels, where at the end of the convolution a $K \times T$ feature vector is produced. This constitutes a single convolutional layer and acts as a feature extractor from the sequence.

Depending on the complexity of the information, CNN can include a whole series of convolution and pooling layers in order to extract more abstract features. The resulting feature maps need to be flattened and concluded by at least one regular fully-connected layer prior to classification. The final layer of the network outputs the predictions.

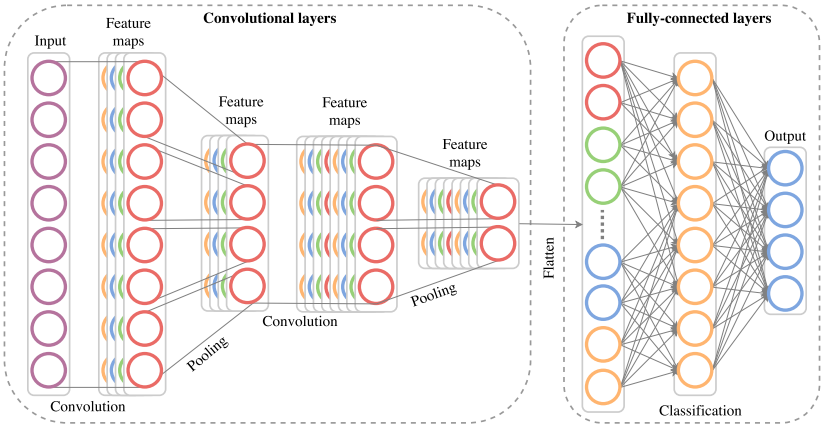
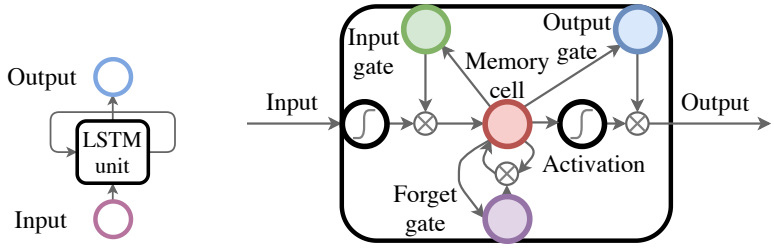


Figure 2.4: Convolutional neural network.

Recurrent Neural Networks and LSTM When the natural form of information is sequential and temporally dependent, traditional feedforward neural networks are not able to effectively learn and infer on these temporal dependencies. Recurrent Neural Networks (RNNs) [93] address the matter by designing networks with feedback connections, storing intermediate states between the time-steps. This ability makes them capable of modeling and inferring on temporal information in the inputs. A RNN can be thought of as multiple copies of the same network, each passing a message to a successor.

Long Short Term Memory (LSTM) [93] networks are an RNN type with stronger memory. It utilizes a so-called LSTM block to model long-term memory, which allows the network to learn longer input sequences. The structure of the LSTM block and the recurrent LSTM layer are shown in Figure 2.5. When modeling temporal dependencies, the copied LSTM layers define which parts of the learned behavior should be maintained and which should be forgotten, via the proper mechanisms like the forget gate shown in purple. The LSTM block processes sequences time step by time step, passing the data through its *memory cells*, and input, output and forget *gates* The block takes three inputs: the current time step, the output and the memory from the previous step. In this manner this single block can process whole sequences of data instead of a single instance and make decisions considering the current input, the previous output and the memorized state.

The memory cell represents the so-called *internal state* of the network. LSTM is able to remove or add information to the cell, regulating these operations by gates. Gates are composed of a sigmoid neural network layer and a pointwise



(a) LSTM neural network

(b) LSTM block

Figure 2.5: LSTM neural network layer and a LSTM block. Recurrence is achieved by feeding the output of a block as an input to the next one. A LSTM network can be "unrolled" to as many layers as there are time steps in the processed sequence.

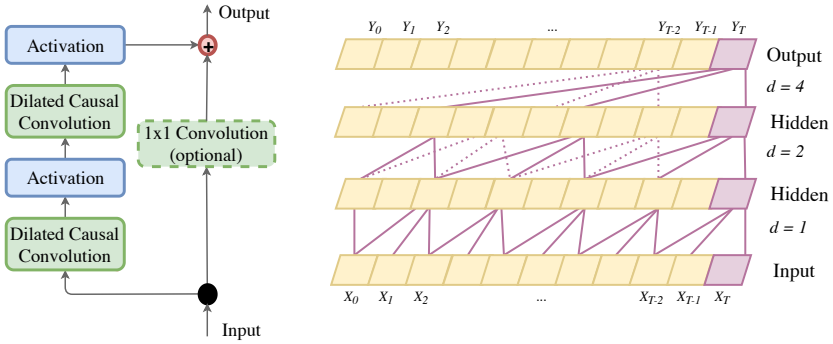
product, and are parameterized by a set of learnable weights. Gates learn to carefully choose whether to let the information through them in order to modify the internal state, to forget information or to produce the output when deemed necessary. The output of a LSTM block is formed by the number of memory units.

LSTM layer's depth depends on the length of processed sequences: due to the feedback connection, they have one layer for every processed time step of a sequence. Such structure can be obtained by unrolling the loop in Figure 2.5. Classification errors are backpropagated through many layers "through time", which limits the training process. Firstly, it significantly slows down training in comparison to the feedforward networks. Secondly, in practice it only allows to backpropagate up to 100-200 layers.

LSTM layers can be stacked with other layers to form deeper networks. The intuition is that higher LSTM layers can capture more abstract concepts. LSTM can also learn temporal data representations before passing them on to a CNN layer, for example.

2.4.3 Temporal Convolutional Neural Network

Recently, the notion of Temporal Convolutional Networks (TCNs), depicted in Figure 2.6 was introduced by Bai et al. [18]. Their main working mechanism is convolution layers, similarly to CNNs, but they have additional architectural characteristics that enable them to interpret sequences better than one-



(a) Temporal convolutional network.

(b) Residual block (k, d).

Figure 2.6: Architectural elements in a TCN: a dilated causal convolution with dilation factors $d = 1, 2, 4$ and filter size $k = 3$ and a TCN residual block.

dimensional CNNs. There are two key components that render them more suitable than 1D CNNs for sequence learning: (a) by employing dilated causal convolutions the issue of future information "leaking" to the past is averted, and (b) by interweaving convolutional and residual layers, the model has better capacity in memorizing longer sequences and distant context. Similarly to recurrent networks, a TCN can take a sequence of any length and map it to an output sequence of the same length.

2.5 Conclusion

In this chapter, we covered the necessary background on DL, with the focus on its usage in applied research and practical deployment. Thereafter, we highlighted the key challenges specific to DL and involved in utilizing DL in cybersecurity and privacy. We supported this discussion with an updated view on relevant developments in fundamental AI research, most of which are yet to be integrated in applied research fields. Finally, we briefly introduced several types of deep neural network architectures that are relevant for the work presented in this dissertation.

Following this chapter, we move on to research studies conducted in the scope of this thesis. The first two chapters revolve around applications of DL in offensive security research, namely its impact on attacks against privacy enhancing technologies.

Part I

Traffic Deanonymization

Chapter 3

Automated Website Fingerprinting through Deep Learning

Preamble

Several studies have shown that the network traffic that is generated by a visit to a website over Tor reveals information specific to the website through the timing and sizes of network packets. By capturing traffic traces between users and their Tor entry guard, a network eavesdropper can leverage this metadata to reveal which website Tor users are visiting. The success of such attacks heavily depends on the particular set of traffic features that are used to construct the fingerprint. Typically, these features are manually engineered and, as such, any change introduced to the Tor network can render these carefully constructed features ineffective. In this chapter, we show that an adversary can automate the feature engineering process, and thus automatically deanonymize Tor traffic by applying our novel method based on deep learning. We collect a dataset comprised of more than three million network traces, which is the largest dataset of web traffic ever used for website fingerprinting, and find that the performance achieved by our deep learning approaches is comparable to known methods which include various research efforts spanning over multiple years. The obtained success rate exceeds 96% for a closed world of 100 websites and 94% for our biggest closed world of 900 classes. In our open world evaluation, the most performant deep learning model is 2% more accurate than the closest state-of-the-art attack [154].

Furthermore, we show that the implicit features automatically learned by our approach are far more resilient to dynamic changes of web content over time. We conclude that the ability to automatically construct the most relevant traffic features and perform accurate traffic recognition makes our deep learning based approach an efficient, flexible and robust technique for website fingerprinting.

This chapter has been previously published as a paper on automated website fingerprinting [170], which has revealed unprecedented advantages of using deep neural networks in traffic deanonymization. Publication of this result brought about a long and fruitful line of research that proposed other deep learning mechanisms for traffic analysis attacks [193, 152, 24, 221, 151], enabling a series of scientific advances. For instance, Sirinam et al. [193] have shown that by significantly increasing the number of layers and parameters of a deep neural network, which increases its learning capacity, they could break one of the practical defenses against website fingerprinting called WTF-PAD [105]. However, the more overparameterized the network, the more challenging it becomes to configure the attack, and the more data is needed for training, which requires continuous updating. Therefore, the practicality of data-hungry deep learning methods in an online environment as a tool at the adversary's disposal is unclear. As a result, several studies focused on reducing the need for the amount of data required for high performance of traffic analysis attacks by exploring the so-called data-limited fingerprinting paradigm. Few-shot learning [194] and generative adversarial networks [151] are examples of highly performant data-limited algorithms for realizing website fingerprinting. Another research direction explores optimal traffic representations, utilizing additional information such as timings [24, 163].

Website fingerprinting is a continuously developing field with many notable contributions. One important line of follow-up work focuses on creating mitigations in Tor [233, 78, 54, 184], which are evaluated against deep learning-based attacks. Another direction considers more realistic evaluations of the privacy leakage by eliminating simplifying assumptions [136] or by considering more suitable performance metrics [222]. Furthermore, Cherubin et al. [42] have recently successfully and responsibly launched website fingerprinting in the online Tor network, showing that the general attack is highly accurate on a small set of sensitive websites but becomes less accurate with the growth of the monitored set.

As was previously mentioned, one of the key contributions of our study is the public release of the largest dataset with website fingerprinting traffic (i.e., client-side Tor traffic of website visits). This dataset has proven to be an important asset for the field. Apart from the open-sourced data formatted in the way it is used in our experiments, the much larger, unprocessed original dataset has also been widely used by independent researchers: specifically, we

received over 50 direct requests to transfer the full dataset. Multiple follow-up studies have used our open-sourced code to reproduce and build upon our results. Today, sharing artifacts in data-driven privacy research is an actively supported practice.

3.1 Introduction

The Onion Router (Tor) is a communication tool that provides anonymity to Internet users. It is an actively developed and well-secured system that ensures the privacy of its users' browsing activities. For this purpose, Tor encrypts the contents and routing information of communications, and relays the encrypted traffic through a randomly assigned route of nodes such that only a single node knows its immediate peers, but never the origin and destination of a communication at the same time. Tor's architecture thus prevents ISPs and local network observers from identifying the websites users visit.

As a result of previous research on Tor privacy, a serious side-channel of Tor network traffic was revealed that allowed a local adversary to infer which websites were visited by a particular user [90]. The identifying information leaks from the communication's metadata, more precisely, from the directions and sizes of encrypted network packets. As this side-channel information is often unique for a specific website, it can be leveraged to form a unique fingerprint, thus allowing network eavesdroppers to reveal which website was visited based on the traffic that it generated.

The feasibility of Website Fingerprinting (WF) attacks on Tor was assessed in a series of studies [155, 224, 104, 154, 225]. In the related works, the attack is treated as a classification problem. This problem is solved by, first, manually engineering features of traffic traces and then classifying these features with state-of-practice machine learning algorithms. Proposed approaches have been shown to achieve a classification accuracy of 91-96% correctly recognized websites [223, 154, 86] in a set of 100 websites with 100 training instances per website. Their works show that finding distinctive features is essential for accurate recognition of websites. Moreover, this tasks can be costly for the adversary as they have to keep up with changes introduced in the network protocol [32, 105, 41]. At that point, the WF research community had not yet investigated the success of an attacker who automates the feature extraction step for classification. This is the key problem that we address in this work.

An essential step of traditional machine learning is feature engineering. Feature engineering is a manual process, based on intuition and expert knowledge, to find a representation of raw data that conveys characteristics that are most

relevant to the learning problem. Feature engineering proved to be even more important than the choice of the specific machine learning algorithm in many applications, including WF [64, 104].

When developing a new WF attack, prior work on WF typically focuses on feature engineering to compose and select the most salient features for website identification. Moreover, these attacks are actually defined by a fixed set of features derived from this process. Thus, these attacks are sensitive to changes in the traffic that would distort those features. In particular, deploying countermeasures in the Tor network that conceal the features is sufficient to defend against such attacks. This enables an arms-race between attacks and defenses: new attacks defeat defenses because they exploit features that had not been considered before and, conversely, new defenses are designed to conceal the features that those attacks exploited.

In this chapter, we propose a novel WF attack based on deep learning. Our attack incorporates automatic feature learning and, thus, it is not defined by a particular feature set. This may be a game-changer in the arms-race between WF attacks and defenses, because the deep learning based attack is designed to be adaptive to any perturbations in the features introduced by defenses. The attack we present in this work is the first automated WF attack and it is at least as effective as the state-of-the-art, manual approaches.

The key contributions of our work are as follows:

- Our study provides the first systematic exploration of state-of-the-art Deep Learning (DL) algorithms applied to WF, namely feedforward, convolutional and recurrent deep neural networks. We design, tune and evaluate three models – Stacked Denoising Autoencoder (SDAE), Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM). Our DL models are capable of *automatically* learning traffic features for website recognition at the expense of using more data. Moreover, we automate the model selection to find the best network hyperparameters. We demonstrate that our DL-based WF attack reaches a high success rate, comparable to the state-of-the-art techniques.

- We reevaluate prior work on our dataset and reproduce their results. We find that state-of-the-art WF approaches benefit from using more training data, similar to DL. As a result of a systematic comparison of our novel DL-based methods to previous WF approaches for the closed and open world settings, we demonstrate comparable recognition results with slight improvements of up to 2%. Furthermore, we show that our DL attack reveals more general and stable website features than the state-of-the-art

methods, which makes them more robust to concept drift caused by highly dynamic web content.

- The dataset collected for the evaluation is the largest WF dataset ever gathered to date. Our closed-world dataset consists of 900 websites, with traffic traces generated by 2,500 visits each. Our open-world dataset is based on 400,000 unknown websites and 200 monitored websites. We made the generated dataset publicly available, allowing researchers to replicate our results and systematically evaluate new (DL) approaches to WF.[1]

This chapter is structured as follows. In Section 3.2, we discuss related work on WF and the use of DL. Section 3.3 presents the threat model and the capabilities an adversary has for WF. The data collection process is outlined in detail in Section 3.4. Section 3.5 provides a reevaluation of state-of-the-art attacks on our dataset and the overall deep learning approach and evaluation. We discuss the results and limitations of our work, as well as opportunities for future research, in Section 3.6. Section 3.7 concludes by summarizing our main findings.

3.2 Background

This section reviews recent related work on Tor WF attacks relying on traditional machine learning algorithms, and the application of deep learning.

Anonymous communications systems such as Tor [57] provide confidentiality of communications and conceal the destination server's address from network eavesdroppers. However, in the last decade, several studies have shown that, under certain conditions, an attacker can identify the destination website only from encrypted and anonymized traffic.

In WF, the adversary collects traffic from their own visits to a set of websites that they are interested in monitoring, visiting each site multiple times. Next, the adversary builds a website *template* or *fingerprint* from the traffic traces collected for that site. The fingerprints are built using a supervised learning method that takes the traffic traces labeled as their corresponding site, extracts a number of features that identify the site and outputs a statistical model that can be used for classification of new, unseen traffic traces. Finally, the attacker applies the classifier on unlabeled traffic traces collected from communications initiated by the victim and makes a guess based on the output of the classifier.

[1] The dataset and implementation can be found on the following URL: `https://tor-wf-dl.distrinet-research.be/`.

To be able to deploy the attack, the adversary must be able to observe the traffic generated by the victim and be able to identify the user (see Section 3.3 for more details on the threat model).

The first WF studies evaluated the effectiveness of the attack against HTTPS [40], encrypted web proxies [202, 92], OpenSSH [118] and VPNs [90] and it was not until 2009 that the first evaluation of a WF attack was performed in Tor [90]. This first attack in Tor was based on a Naive Bayes classifier and the features were the frequency distributions of packet lengths [90]. Even though their evaluation showed the attack achieved an average accuracy of only 3%, the attack was improved by Panchenko et al. using a Support Vector Machine (SVM) [155]. In addition, Panchenko et al. added new features that were exploiting the distinctive *burstiness* of traffic and increased the accuracy of the attack to more than 50%.

These works were succeeded by a series of studies that claimed to boost the attacks and presented attacks with more than 90% success rates. First, Cai et al. [33] used an SVM with a custom kernel based on an edit-distance and achieved more than 86% accuracy for 100 sites. The edit distance allowed for delete and transpose operations, that are supposed to capture drop and retransmission of packets respectively. Following a similar approach, Wang and Goldberg [224] experimented with several custom edit distances and improved Cai et al.'s attack to 91% accuracy for the same dataset.

However, these evaluations have been criticized for making unrealistic assumptions on the experimental settings that give an unfair advantage to the adversary compared to real attack settings [104]. For instance, they evaluated the attacks on small datasets and considered adversaries who can perfectly parse the traffic generated by a web-page visit from all the traffic that blends into the Tor network. Furthermore, they assume users browse pages sequentially on one single browser tab and never interrupt an ongoing page-load. Recent research has developed new techniques to overcome some of these assumptions, suggesting that the attacks may be more practical than previously expected [225].

The three most recent attacks in the literature outperform all the attacks described above and, for this reason, we have selected them to compare with our DL-based attack. Each attack uses a different classification model and feature sets and work as follows:

Wang-kNN [223]: this attack is based on a k-Nearest Neighbors (k-NN) classifier with more than 3,000 traffic features. This large number of features is obtained by varying the parameters of a set of fewer feature *families*. For instance, the number of outgoing packets in spans of X packets and the lengths

of the Y packets in the same direction. In order to mitigate the curse of dimensionality, they proposed to weigh the features of a custom distance metric, minimizing the distance among traffic samples that belong to the same site. Their results show that this attack achieves 90% to 95% accuracy in a closed world of 100 websites [223].

CUMUL [154]: CUMUL is based on an SVM with a Radial Basis Function (RBF) kernel. CUMUL uses the cumulative sum of packet lengths to derive the features for the SVM. The cumulative sum is computed by adding the lengths of outgoing packets and subtracting the lengths of incoming packets. However, since the RBF kernel, in contrast to the aforementioned edit-distance based SVM kernel, expects feature vectors to have the same dimension, they interpolated 100 points from the cumulative sums. Furthermore, they prepend the total incoming and outgoing number of packets and bytes. As a result, they ended up with 104 features to represent a traffic instance. Their evaluations demonstrate an attack success that ranges between 90% and 93% for 100 websites. It is worth mentioning that their dataset is the most realistic up to date, including *inner* pages of sites that have spikes of popularity such as Google searches or Twitter links. Despite the high success rate of their attack, the authors conclude that the WF attack does not scale when applied in a real-world setting, as an adversary would need to train the classifier on a large fraction of all websites.

k-Fingerprinting (k-FP) [86]: Hayes and Danezis's k-FP attack is based on Random Forests (RFs). Random Forests are ensembles of decision trees that are randomized and averaged so that they can generalize better than simple decision trees. Their feature sets include 175 features developed from features available in prior work, as well as timing features that had not been considered before, such as the number of packets per second. The random forest is not used to classify but as a way to transform these features into a different feature space: they use the leafs of the random forest to encode a new representation of the sites they intent to detect that is relative to all the other sites in their training set. Next, the new representation of the data is fed to a k-NN classifier for the actual classification. Their results show that this attack is as effective as CUMUL and achieves similar accuracy scores for the same number of sites.

All these attacks have selected their features mostly based on expertise and their technical knowledge on how Tor and the HTTP protocol work and interact with each other. As a result of manual feature engineering and standard feature selection, each proposed attack can be represented by a set of fingerprinting features. It is still unknown whether WF can be successfully deployed through automatic feature engineering based on implicit uninterpretable traffic features.

To the best of our knowledge, the only research that successfully applies deep learning to a similar problem is the network protocol recognition on encrypted traffic with a SDAE done by Wang [229]. His approach achieves a 90% recognition rate, which is a promising indicator for deep learning application to anonymized traffic.

The first effort to apply a DL-based approach to WF was made by Abe and Goto [1], where they evaluated a SDAE on the Wang-kNN's dataset. Their classifiers do not outperform the state-of-the-art, but nevertheless achieve a convincing 88% on a closed world of 100 classes. It is fair to assume that the lower performance is due to the lack of a sufficient amount of training data for a deep neural network, which, as we confirm later in this chapter, is essential for the deep learning performance. Moreover, the work does not assess applicability of other deep learning algorithms to the problem. In this chapter, we explore three deep learning methods when applied to a significantly larger closed world of varying sizes, trained on sufficient amounts of data and evaluated in context of dynamic changes of web content over time. We provide a more extensive tuning of the DL-based attacks and finally achieve a similar accuracy to the state-of-the-art WF attacks.

3.3 Threat Model

In this chapter, we consider an adversary similar to the one considered in prior work in WF, namely a *passive* and *local* network-level adversary. Figure 3.1 shows an overview of this WF scenario. A passive adversary only records network packets transmitted during the communication and may not modify them or cause them to drop, and may not insert new packets into the stream of packets. A local adversary has a limited view of the network. In particular, in Tor, such an adversary typically owns the entry node to the Tor network (also known as *entry guard*), or has access to the link between the client and the

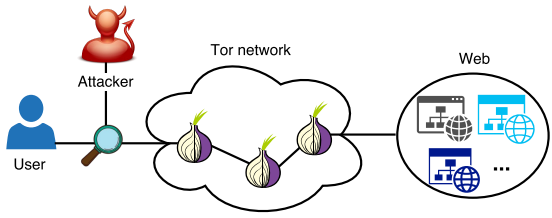


Figure 3.1: The client visits a website over the Tor network. The adversary can observe the (encrypted) traffic between the client and the *entry* to the Tor network.

entry. Examples of entities that have this level of visibility range from Internet Service Providers (ISPs), Autonomous Systems (ASs) or even local network administrators. Note that an adversary that owns the entry guard can decrypt the first layer of encryption and access Tor protocol messages. In this work, we assume an ISP-level adversary that collects traffic at the TCP layer and infers the cells from TCP packets [224]. Obviously, all work on WF assumes the adversary cannot decrypt the encryption provided by Tor, as message contents would immediately reveal the identity of the website.

In the WF literature, it is common for the evaluation of the attack to assume a closed world of websites. This means that the user can only visit pages that the adversary has been able to train on. This assumption, commonly known as the *closed-world* assumption, has been deemed unrealistic [155] as the size of the Web is so large that an adversary can only train on a tiny fraction of the Web. For this reason, many studies have also evaluated the more realistic *open world*, where the user is allowed to visit pages that the adversary has not trained on. The closed world is still useful to compare existing attacks and defenses. In this study, we evaluated both the closed world and the open world.

3.4 Data Collection

One of the prerequisites for deep learning is an abundance of training data required to learn the underlying patterns. Processing sufficient amounts of representative data enables the deep neural network to not only precisely reveal the identifying features but also generalize better to unseen test instances. In prior work on WF in the context of Tor, the datasets that were collected are relatively limited in size, both in terms of classes (i.e. the number of unique websites) as well as instances (i.e. the number of traffic traces per website). To properly evaluate our proposed deep learning approach and explore how existing models can benefit from extra training data, we used a distributed setup to collect various new datasets that accommodate these requirements.

3.4.1 Data Collection Methodology

For the data collection process, we used 15 virtual machines on our OpenStack-based private cloud environment. Each VM was provisioned with 4 CPUs and 4GB of RAM. To each VM, 16 worker threads were assigned, which each had their separate `tor` process (version 0.2.8.11). Page-visit tasks, consisting of starting the Tor browser (version 6.5) and loading the target web page, were then distributed among the 240 concurrent worker threads. Web pages were

given 285 seconds to load, before the browser was killed and the visit marked as invalid. Upon loading the page, it was left open for an additional 10 seconds, after which the browser was closed and any profile information was removed.

By leveraging network namespaces and `tcpdump`, we isolated and captured the traffic of each `tor` process. For the purpose of sanity checks and validation, information on the Tor circuit that was used for the page visit is also recorded. Due to storage constraints, and since the packet payloads are encrypted and thus do not have value for the adversary, we extract metadata from the traffic trace and discard the encrypted payload. More precisely, we capture the following metadata: (i) the timing information, (ii) the direction and (iii) the size of the TCP packet.

To obtain a useful traffic trace representation, we follow the approach proposed by Wang and Goldberg [224] which works with Tor *cells* instead of original TCP/IP packets. Tor cells are basic units of data transfer in Tor with a fixed size (512 bytes), which are encrypted in TLS records and encapsulated in TCP/IP packets. Since the adversary operates at a TCP level, they can extract Tor cells from the captured TCP traffic. Wang and Goldberg propose to first parse TCP/IP data to reconstruct full TLS records with their lengths, and then infer the number of underlying cells by rounding each length down to the closest multiple of 512 and dividing it by 512. Our final representation of the traffic trace is, hence, an ordered sequence of cells, where each cell is encoded as 1 when transmitted from the client to the website and as -1 when captured in the opposite direction. For instance, a sequence of three TLS records of lengths 544, -1088, and 1088 bytes would be represented as 1, -1, -1, 1, 1.

It should be noted that, in contrast to prior work [224], the Tor entry guard node was not pinned over the course of our experiments. The reason for this is twofold. First, compared to prior data collection, we use significantly more concurrent processes. If the same entry guard would be used by the 240 browser instances, this could overload the entry guard, possibly affecting the network traces. Second, by using a variety of entry guards, the trained models are agnostic to the intrinsics of a specific entry guard. This means that the model of the adversary is not only applicable in a targeted attack on a single victim, but can be launched against any Tor user.

3.4.2 Datasets

Since the WF adversary's goals might vary widely and as there are no statistics about which pages Tor users browse, there can be no definitive set of sensitive websites for WF research. Moreover, since we aim to compare various approaches with each other, the actual choice of websites is not essential as long as it is

consistent. The list of websites we chose for our evaluation comes from the Alexa Top Sites service, the source widely used in prior research on Tor.

In total, we evaluate our deep learning approach in comparison with traditional methods on three different datasets. This section details how these datasets were chosen and obtained.

Closed World

For the dataset under the *closed world* assumption, we collected up to 3,000 network traces for visits to the homepage of the 1,200 most popular websites according to Alexa. The list of popular websites was first filtered to remove duplicate entries that only differ in the TLD, e.g. in the case of `google.com` and `google.de`, only the former was included in the list. Data for these 1,200 websites was collected in four iterations, consisting of 300 websites each. An iteration was again split up into 30 batches, with each batch performing 100 network traces per websites. After each batch, the 240 `tor` processes were restarted and data directories were removed, forcing new circuits to be built with (new) randomly selected entry guards. Network traces for each of the four iterations were collected over approximately 14 days per group, starting from January 2017.

After collecting data on the 3.6 million page visits, we filtered out invalid entries, which were due to a timeout, or a crash of the browser or Selenium driver. Websites with a high number of invalid page visits were removed from our dataset. Additionally, using the similarity hash of the web page's HTML content [38] and the perceptual hash of the screenshot [30], we detected and excluded websites with exactly the same content. Moreover, we filtered out websites that had no content, denied all requests coming from Tor, or showed a CAPTCHA for every visit. Finally, we balanced the dataset to ensure the uniform distribution of instances across different sites by fixing the same number of traces for every site. After this filtering process, our biggest closed world dataset consists of 900 websites, with 2,500 valid network traces each. In the remainder of the text, we refer to this dataset as CW_{900}. Similarly, for datasets that are composed of a subset of this one we use a corresponding representation: the datasets for the Alexa top 100, 200 and 500 websites are referred to as CW_{100}, CW_{200} and CW_{500} accordingly.

Revisit over Time

For the top 200 websites, we obtained additional periodic measurements. More precisely, for these websites we collected 100 test network traces per website 3 days, 10 days, 4 weeks, 6 weeks and 8 weeks after the end of the initial data collection for these 200 websites. Each test set is collected within one day. As a result, our *revisit-over-time* dataset provides 500 network traces for each of the top 200 websites collected over a 2-month period (CW_{200} was collected over 2 weeks).

Open World

Since the *open world* data is only used for testing purposes (which differs from some of the open world evaluations), we collected only a single instance for each page in the open world. In total, we collected network traces for the top 400,000 of Alexa websites.

We collected additional 2,000 test traces for each website of the monitored closed world CW_{200} (400,000 instances in total). As a result, we conduct the open world evaluation on 800,000 test traffic traces, half from the closed world and half from the open world (a 4-fold increase compared to the largest dataset considered in prior work [86, 154]). We provide the motivation for this experimental setting in Section 3.5.2.

3.4.3 Ethical Considerations and Data Access

For our data collection experiments, we performed around 4 million page visits over Tor. It is highly unlikely that this had any impact on the top websites, which each receive multiple millions of requests every day. We consider the impact on the Tor network to be limited as well: The Tor Project estimates that during the time we performed our experiments, approximately 2 million clients were concurrently connected to the Tor network. As such, the 240 clients we used are only a minor fraction of the total number of active clients. Furthermore, we made the data publicly available, allowing other researchers to evaluate other approaches without having to collect new data samples.

3.5 Evaluation

In this section, we conduct a reevaluation of the state-of-the-art WF methods discussed in the related work of Section 3.2 to confirm their reproducibility on our dataset. We then evaluate the proposed attacks based on the three chosen DL algorithms and compare them to the previously known techniques.

3.5.1 Reevaluation of the State-of-the-Art

We aim to enable a systematic comparison between our work and that of Wang et al. [223], Panchenko et al. [154] and Hayes et al. [86], not only to guarantee a fair assessment by evaluating on new data, but also to analyze (i) the practical feasibility of the attack on a significantly larger set of websites, (ii) the impact of collecting more instances or traces per website on the classification accuracy, and (iii) the resilience of trained models to concept drift with a growing time gap between training and testing.

The goal of the first closed world experiment is to confirm whether we can reproduce the three WF attacks of prior work [223, 154, 86] and to assert whether we obtain similar classification results as those reported by the respective authors, but on a different training and testing dataset similar in size. We reuse the original implementation of the authors to carry out the feature extraction and subsequently execute the training and testing steps. All results reported in this section are computed via 10-fold cross-validation.

The following results were obtained on a Dell PowerEdge R620 server with 2x Intel Xeon E5-2650 CPUs, 64GB of memory and 8 cores on each CPU with hyperthreading, resulting in 32 cores in total each running at 2GHz. Wang's k-NN based attack ran on a single core as the stochastic gradient descent method to find the best weights for k-NN classification could not be parallelized without sacrificing some classification accuracy. Panchenko's CUMUL attack trains an SVM model which requires a grid search to find the best C and γ parameter combination for the RBF kernel. As the native libSVM library is not multi-core enabled, the parameter combination tests ran as parallel processes each on a single core, with the time reported being the one of the slowest C and γ parameter combination test.

Figure 3.2 shows the closed-world classification accuracy obtained through cross-fold validation for the three traditional WF attacks on a CW_{100} dataset with 100 traces per website. For the same set of website instances, the k-NN algorithm of Wang et al. reports a classification accuracy of 92.87% on our new data set, whereas the CUMUL algorithm of Panchenko et al. and the k-FP

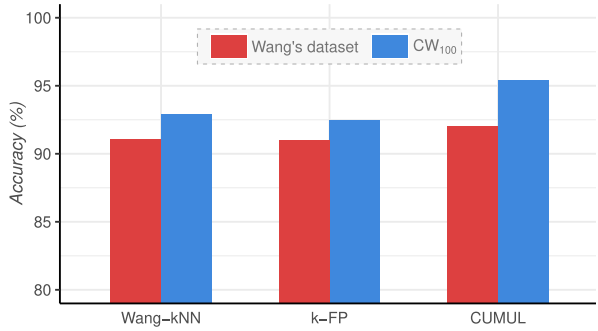


Figure 3.2: Re-evaluation of traditional WF attacks on new data.

attack by Hayes et al. respectively report accuracy results of 95.43% and 92.47%. The obtained results are in line with those originally reported by the authors themselves albeit on other data sets. For this particular setup, the CUMUL WF attack turned out to be the most accurate.

In the second experiment, we evaluate the same traditional methods on 100 websites, but with a growing number of traces per website, to investigate whether the classification accuracy improves significantly when provided with more training data and whether one WF attack method is consistently better than another.

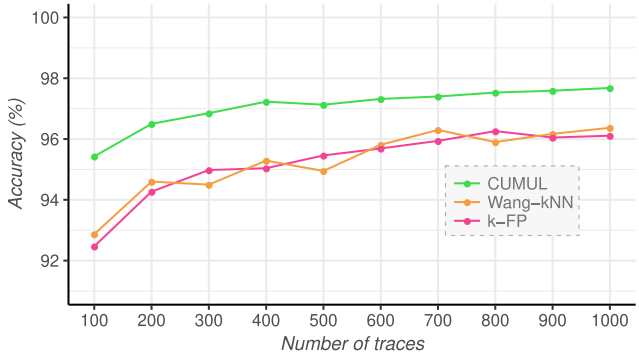


Figure 3.3: Impact on the classification accuracy for a growing number of website traces.

In Figure 3.3, we depict the classification accuracy in a closed world experiment where the number of website instances grows from 100 to 1,000 traces. Our results show that the CUMUL attack consistently outperforms the two other methods. For all methods, the improvement becomes less evident after about

300 website traces. Another interesting observation is that each WF attack − when given sufficient training data − converges to a classification accuracy of approximately 96-98%. However, we experienced scalability issues with the k-NN based attack by Wang et al., given that the classification running times were at least an order of magnitude higher than those of the CUMUL and k-FP attacks.

In a third experiment, we assess how the classification accuracy drops when the number of websites increases for a fixed number of training instances. Given that the CUMUL attack consistently outperformed the other two methods on our dataset, and was superior in resource consumption, we only report the results for CUMUL. We reevaluate the CUMUL classifier on our closed worlds CW_{100}, CW_{200}, CW_{500} and CW_{900} with a fixed number of traffic traces: 300 per website.

Table 3.1 illustrates that the CUMUL attack obtains a reasonable 92.73% 10-fold cross-validation accuracy for 900 websites using 300 instances each, and a parameter combination of $log_2(C) = 21$ and $log_2(\gamma) = 5$. In general, we observe that the performance degrades gradually with a growing size of the closed world. Moreover, doubling the initial number of instances gives an advantage of up to 2%, while the amounts higher than 300 stop providing any significant improvement. The biggest weakness is that for each experiment one must execute the grid search to ensure the best classification results, and certain parameter combination tests take a long time to converge with no guarantee of a gain in accuracy.

Table 3.1: CUMUL accuracy for a growing closed world (with 100 traces per website, 300 traces, and the best achieved accuracy for a varying number of traces).

Dataset	**CUMUL (100tr)**	**CUMUL (300tr)**	**CUMUL (best)**
CW_{100}	95.43 %	96.85 %	97.68 % (2000 tr)
CW_{200}	93.58 %	95.93 %	97.07 % (2000 tr)
CW_{500}	92.30 %	94.22 %	95.73 % (1000 tr)
CW_{900}	89.82 %	92.73 %	92.73 % (300 tr)

Table 3.2 gives an overview of the running times (in minutes) to find the best C and γ parameter values for the RBF kernel. We aborted those experiments where the grid search took more than four days to complete. While there is a trend of increasing values for these parameters with a growing number of websites and instances, we could not find a strong correlation that would enable us to eliminate the grid search altogether.

As a result, we choose CUMUL as the reference point for comparing our

Table 3.2: Time required to find optimal RBF parameter values for C and γ for SVM based classification.

Traces	CW_{100}	CW_{200}	CW_{500}	CW_{900}
100	3 min	8 min	139 min	771 min
200	10 min	48 min	684 min	3027 min
300	19 min	99 min	1230 min	4031 min
400	29 min	134 min	1490 min	$> 6000^{\dagger}$ min
500	34 min	169 min	1541 min	$> 6000^{\dagger}$ min
1000	41 min	844 min	5016 min	$> 6000^{\dagger}$ min
2000	41 min	844 min	5016 min	$> 6000^{\dagger}$ min

†Aborted experiments.

proposed method with the state-of-the art. This decision is driven by the fact that CUMUL performed the best on our closed worlds, and proved to be more practically feasible. We acknowledge that the k-FP attack has the potential to work better in our open world evaluation. However, over the course of our scalability experiments, k-FP did not scale to 50,000 training instances. The experiment consumed more than 64GB memory and took longer than the allocated 4 days, and thus was aborted.With our open world datasets consisting of 800,000 instances (and 400,000 training instances), such high resource consumption demands strongly limit large scale evaluation. CUMUL on the other hand scales up to 400,000 training instances. Therefore, we further evaluate our DL-based approach in comparison to CUMUL, which outperformed the other traditional WF techniques and which was practically feasible on a larger scale.

3.5.2 Deep Learning for Website Fingerprinting

Here we provide a detailed outline of our DL-based methodology. DL provides a broad set of powerful machine learning techniques with deep architectures. Deep Neural Networks (DNNs), which underlie DL, exploit many layers of non-linear mathematical data transformations for automatic hierarchical feature extraction and selection. DNN demonstrate a superior ability of feature learning for solving a wide variety of tasks. In this study we apply three major types of DNNs to WF: a feedforward SDAE, a convolutional CNN and a recurrent LSTM.

Problem Definition

In our proposed method, we follow prior work and formulate WF as a classification problem. Namely, we perform a supervised multinomial classification, where we train a classifier on a set of labeled instances and test the classifier by assigning a label out of a set of multiple possible labels to each unlabeled instance. In WF, a traffic trace t captured from a single visit to a website is an instance of the form $(\mathbf{f_t}, c_t)$, where $\mathbf{f_t}$ is the feature vector of the traffic trace and c_t is the class label that corresponds to the website that generated this traffic. Assuming a closed world of N possible websites, label c_t belongs to the set $\{0, 1, \ldots, N-1\}$. As such, we state the WF problem as follows: assign a class label to each anonymous traffic trace in a dataset based on its features.

The classifiers used in related work successfully solved this problem by carefully constructing feature vectors, as described in Section 3.2. Our proposed classifier, based on a DNN, integrates feature learning within the training process, enabling it to classify traffic traces simply based on their initial representation. Thus, for a DL classifier, the form of the input instance changes to $(\mathbf{r_t}, c_t)$, where $\mathbf{r_t}$ is a raw representation of a traffic trace that can be interpreted by a neural network.

In essence, we represent a traffic trace as a sequence of successive Tor cells that form the communication between the target user and the visited website. As a result, an input instance of our DNN-based classifier is a series of 1 and -1 of variable length, based on which the model performs feature learning and website recognition. Our choice of this format is also supported by the fact that neural networks generally work with real numbers from the compact interval $[-1, 1]$ due to the nature of the mathematical operations they perform. Moreover, by providing the input data in such a format, we avoid having to rescale and/or normalize the values and thus mitigate a possible information loss coupled with the preprocessing step.

Out of all existing types of DNNs and corresponding DL algorithms, we evaluate three major types of neural networks: feedforward, convolutional and recurrent. We choose to apply the models that provide the capabilities and architectural characteristics to perform the task of automated feature extraction and to benefit from the nature of our input data. We refer to Chapter 2 for a more elaborate and in-depth discussion on the DL algorithms, which we consider to be conceptually the most well-suited for the WF task at hand. Appendix A.1 illustrates more exactly the chosen DL algorithms in application to website fingerprinting.

The first DNN we apply is a classifier called Stacked Denoising Autoencoder

(SDAE) – a deep feedforward neural network composed of Denoising Autoencoders (DAEs). An Autoencoder (AE) is a feedforward network specifically designed for feature learning through dimensionality reduction. Stacking multiple AEs as building blocks to form a deep model allows for hierarchical extraction of the most salient features of the input data and performing classification based on the derived features, which makes SDAE a promising model for our WF problem.

The next proposed DNN is a Convolutional Neural Network (CNN) – a classifier built on a series of convolutional layers. Convolutional layers are also used for feature extraction, starting with low-level features at the first layer and building up to more abstract concepts going deeper in the network. CNN's methodology for achieving that differs from that of SDAE. Convolutional layers learn numerous filters that reveal regions in the input data containing specific characteristics. These input instances are then downsampled with the special regions preserved. In such a way the CNN searches for the most important features to base the classification on. Furthermore, while SDAE has to be pretrained block by block, CNN requires minimum preprocessing.

The final chosen DNN is yet another type of a neural network, very different in its fundamental properties from the first two. A classifier called Long Short Term Memory (LSTM) is a special type of a recurrent neural network that has enhanced memorization capabilities. Its design allows for learning long-term dependencies in data, enabling the classifier to interpret time series. Our input traffic traces are essentially time series of Tor cells, and temporal dynamics in these series are expected to be highly revealing of the contained website fingerprint, thus the choice of the model.

We used `Keras`[43] with `Theano`[208] backend for the implementation of the DNN classifiers. The source code is publicly available on the following webpage: `https://tor-wf-dl.distrinet-research.be/`.

Hyperparameter Tuning and Model Selection

The adversary has to empirically select a DNN model to apply for WF. For that, the adversary should tune the hyperparameters of the DNN to achieve the best classification performance and, at the same time, enhance its capabilities to generalize well to unseen traffic traces.

Performing an automatic search of the best hyperparameters – be that an exhaustive grid search, a random search or another search algorithm – is highly effective but computationally expensive at the same time. In our work, we evaluate the DL algorithms applied to WF by performing semi-

automatic hyperparameter tuning, where we exploit the knowledge of each hyperparameter's impact. Namely, the main strategy is as follows:

- The adversary chooses a *representative subsample* of the given dataset and splits it *randomly* into training set, validation set and test set in the following proportion: 90% - 5% - 5%
- Next, the adversary defines the limits of the model capacity based on the amount of available training data. On the one hand, the model has to be expressed with a sufficient number of parameters in order to be able to learn the problem. On the other hand, there has to be *much fewer* trainable parameters than available training instances in order to avoid overfitting. The model's capacity is defined through its structure and hyperparameters, different for each DNN. The adversary has to define the search spaces for each hyperparameter.
- In our evaluation a special form of Bayesian optimization is applied for hyperparameter tuning, specifically a Tree of Parzen Estimators (TPE) [21] implemented in the `hyperopt` library. Through this algorithm the adversary automates the tuning process within previously defined search spaces.
- The optimization algorithm returns the best combination of values and the network structure based on the test results. If the adversary finds the model's test performance satisfactory, they select this model. Otherwise, they adjust the search spaces and repeats the tuning procedure.
- Finally, the adversary builds and initializes the selected learning model and applies it to the *whole* dataset to deploy the actual WF attack.

Traditional machine learning methods used for WF in the related work (such as SVM, k-NN and RF, as presented in Section 3.2) also require hyperparameter tuning, but on a smaller scale than DL. Nevertheless, tuning the parameters of the DL model becomes even more feasible in comparison to traditional models due to the parallelism of DL algorithms. As learning algorithms of neural networks are inherently parallel, Graphical Processing Units (GPUs) can take advantage of this characteristic. Performing hyperparameter tuning on GPUs compromises for intense computational requirements allows for rapid feedback of the model. For our DL experiments we use two Nvidia GeForce GTX 1080 GPUs with 8GB memory and 2560 cores each and one TITAN Xp with 12GB memory and 3840 cores to accommodate parallelized training of the DNNs. The training runtime reported in this chapter should therefore be interpreted in association with said platforms.

Table 3.3 includes the list and the values of the hyperparameters we tuned, together with the corresponding intervals within which we vary the values. Each hyperparameter controls a certain aspect of the DL algorithm: architecture (structural complexity of the network), learning (the training process) and regularization (constraint of the learning capabilities applied order to avoid *overfitting*, which occurs when the model memorizes the training data instead of learning from it). Note that in order to reduce the search space, we limited our models to the same learning and regularization parameters for each network layer.

Table 3.3: Tuned hyperparameters of the selected DL models.

	SDAE		*CNN*		*LSTM*	
Hyperparameter	**Value**	**Space**	**Value**	**Space**	**Value**	**Space**
optimizer	SGD	SGD, Adam RMSProp	RMSProp	SGD, Adam RMSProp	RMSProp	SGD, Adam RMSProp
learning rate	0.001	0.0001 .. 0.1	0.0011	0.0009 .. 0.0025	0.001	0.0001 .. 0.1
decay	0.0	0.0 .. 0.9	0.0	0.0 .. 0.9	0.0	0.0 .. 0.9
batch size	32	8 .. 256	256	8 .. 256	128	32 .. 256
training epochs	$\leq$30	1 .. 100	3-6	1 .. 20	$\leq$50	1 .. 100
number of layers	5	3 .. 7	8	6 .. 10	4	3 .. 6
input units	5000	200 .. 5000	3000	200 .. 5000	150	70 .. 1000
hidden layers units	1000, 500, 300	200 .. 3000	—	—	64, 64 / 128, 128	64 .. 256
dropout	0.1	0.0 .. 0.5	0.1	0.0 .. 0.5	0.22	0.0 .. 0.5
activation	`tanh`	`tanh`, `sigmoid`, `relu`	`relu`	`tanh`, `relu`	`tanh`	`tanh`, `sigmoid`, `relu`
pretraining optimizer	SGD	SGD, Adam	—	—	—	—
pretraining learning rate	0.1	0.01 .. 0.1	—	—	—	—
kernels	—	—	32	4 .. 128	—	—
kernel size	—	—	5	2 .. 50	—	—
pool size	—	—	4	2 .. 16	—	—

The adversary is supposed to select the DL-based model *once* given a sample crawled for a desired closed world of websites. Similarly, we perform the model selection on the CW_{100} dataset, as defined in Section 3.4, in order to limit the computational requirements. Given a proper tuning procedure and a sufficiently large number of training instances for each class, the chosen model is expected to learn the problem (learn to extract the fingerprints), and at the same time

generalize well to other closed world datasets. In fact, an adversary capable of crawling large amounts of data can compensate on hyperparameter tuning.

The final selected models of SDAE, CNN and LSTM used for evaluation are described in Table 3.3. The number of LSTM units has to be adjusted for the bigger closed worlds to increase expressive capacity. Note that due to the LSTM's backpropagation-through-time constraints, we have to trim the traffic traces to the first 150 Tor cells.

Further in this subsection we present the experimental results of the DL-based WF attack on the crawled dataset. Namely, we evaluate the three chosen DNNs on the closed worlds of various sizes and on the open world. We also assess their generalization capabilities by testing their resilience to concept drift on data periodically collected over 2 months. Furthermore, we compare results to CUMUL, being the most accurate traditional WF method.

Closed World Evaluation

In this study, we evaluate the SDAE, CNN and LSTM networks on four closed worlds of different sizes, namely CW_{100}, CW_{200}, CW_{500} and CW_{900}. We use the models selected by performing hyperparameter tuning on the CW_{100} dataset, according to the aforementioned methodology. To ensure the reliability of our experiments, we estimate the models' performance by conducting a 10-fold cross-validation on each dataset. We use two performance metrics to evaluate and compare the models with each other: the test *accuracy* (classification success rate, which needs to be maximized) and the test *loss* (a cost function earlier introduced in Chapter 2, it reflects the significance of classification errors made by the model, namely the categorical cross-entropy, that needs to be minimized).

The aspect that had the greatest impact on the performance over the course of our experiments was the amount of training data (i.e., the number of traffic traces for each website), which is in line with our expectations and justifies the extensive data collection. Indeed, for every closed world experiment, we observed significant improvements for a growing number of traces. One example of this trend is given in Figure 3.4 for the CW_{100} dataset, where we vary the number of instances from 100 to all available 2,500 per class. The Table 3.4 reports on the actual metrics' values and the corresponding runtimes.

Table 3.4: Accuracy (Acc), loss, and runtime (RT; in minutes) of the deep learning models SDAE, CNN, and LSTM for CW_{100} and a growing number of traces.

	SDAE			*CNN*			*LSTM*		
Traces	**Acc.**	**Loss**	**RT**	**Acc.**	**Loss**	**RT**	**Acc.**	**Loss**	**RT**
100	85.00 %	0.5902	0 '	81.25 %	0.8276	0 '	40.60 %	2.2132	9 '
200	87.30 %	0.5252	1 '	86.63 %	0.5793	0.5 '	57.30 %	1.5471	17 '
500	91.34 %	0.3576	1 '	91.43 %	0.3877	1 '	79.54 %	0.7848	40 '
1000	92.64 %	0.2950	2 '	94.72 %	0.2545	1.5 '	91.63 %	0.3555	63 '
1500	94.49 %	0.2314	4 '	95.95 %	0.1855	2 '	91.93 %	0.3055	66 '
2000	95.17 %	0.1955	6 '	96.14 %	0.1699	3 '	93.98 %	0.3277	67 '
2500	95.46 %	0.1968	7 '	96.26 %	0.1784	5 '	94.02 %	0.3204	76 '

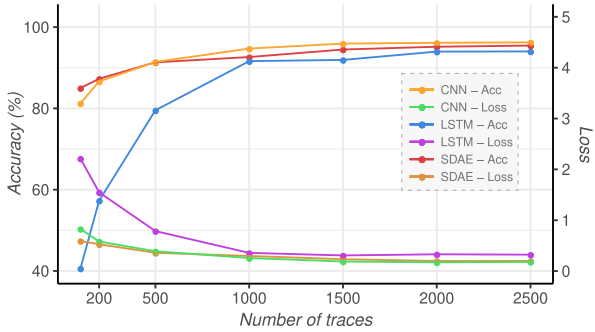


Figure 3.4: Accuracy, loss, and evaluation time of the DL models (SDAE, CNN, LSTM) for CW_{100} and a growing number of traces.

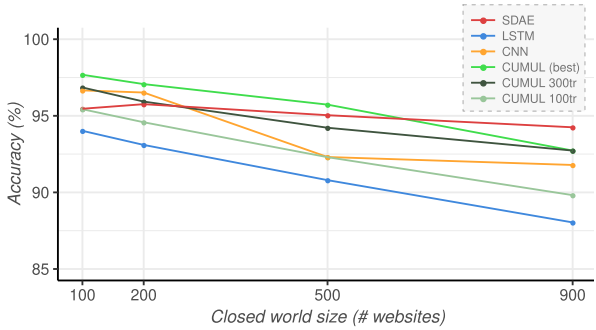


Figure 3.5: DL (SDAE, CNN, LSTM) vs. CUMUL for a growing size of the closed world from 100 to 900 websites.

Table 3.5: Accuracy (Acc), loss, and runtime (RT; in minutes) of the deep learning models SDAE, CNN, and LSTM for each closed world and 2,500 traces.

	SDAE			*CNN*			*LSTM*		
Dataset	**Acc.**	**Loss**	**RT**	**Acc.**	**Loss**	**RT**	**Acc.**	**Loss**	**RT**
CW_{100}	95.46 %	0.1968	7 '	**96.66 %**	0.1699	5 '	94.02 %	0.3204	76 '
CW_{200}	95.76 %	0.1822	14 '	**96.52 %**	0.1774	8 '	93.10 %	0.3292	91 '
CW_{500}	**95.04 %**	0.2243	34 '	92.31 %	0.3732	12 '	90.80 %	0.3163	257 '
CW_{900}	**94.25 %**	0.2530	52 '	91.79 %	0.4278	20 '	88.04 %	0.3601	276 '

First and foremost, from these results we can confirm the *feasibility of the WF attack based on a DL approach with automatic feature learning.* We observe how classification accuracy and loss function gradually improve for all models, in the end reaching the 95.46, 96.66 and 94.02% success rate for SDAE, CNN and LSTM model accordingly. These results are comparable to the ones achieved by traditional approaches in Section 3.5.1.

If we compare the three DNNs with each other, we observe that the SDAE and CNN networks consistently perform better than the LSTM in terms of classification accuracy, with CNN being the most performant. Nevertheless, knowing that the LSTM classifies traffic traces based solely on their first 150 Tor cells (compared to the SDAE and CNN that use up to 5,000 and 3,000 cells from each trace), the achieved performance still appears promising. Our interpretation is that even a small part of the traffic trace is sufficient for website recognition up to 94% accuracy when deploying a model that is able to exploit temporal dependencies of the input sequence. Notably, LSTM performs much poorer when trained on fewer traffic traces than SDAE and CNN, but later gains comparable recognition rate at 1000 training instances per class.

Next, we assess whether the selected DL models tuned on CW_{100} perform similarly when applied to the larger datasets: CW_{200}, CW_{500} and CW_{900}. The results of the DL-based WF for all closed world datasets are presented in Table 3.5, expressed in classification accuracy, loss function and runtime. The time reported in the table is the average time required to build, train and evaluate a model. We observe that for larger closed worlds the performance of the three DL models gradually decreases following a similar trend. The closed world evaluation results remain comparable to CUMUL's results presented in Table 3.1 in the previous subsection. Figure 3.5 compares the DL-based methods to CUMUL. This comparison illustrates that our DL-based attack can indeed successfully learn the fingerprinting features in an automated manner. Furthermore, the training method itself is highly parallelizable on GPU hardware resulting in a faster and therefore more practical closed world WF attack.

The presented experiments on the closed world reflect the model's ability to classify traffic traces that are collected at the same moment as the training data. Even though we prove that such a WF attack is possible, we do not address the question of eliciting the concrete data features that the models take decisions upon. In other words, just based on this experiment, we cannot certainly infer if the DNN reveals the actual website fingerprint for deanonymization, or also learns occasional dynamics in the traffic data instead that just happens to enable recognition. The next experiment is intended to reveal how well our DNNs are able to extract the fingerprint and generalize to new data.

Concept Drift Evaluation

The challenge of recognizing traffic traces collected over time was first addressed by Juarez et al. [104]. They showed that classification accuracy drops drastically when testing the model on traffic captured 10 days after training. This time effect is explained by constant content changes of the websites, which of course may affect the identifying fingerprints. Another possible reason for the performance drop is that the classifier trained and evaluated at one moment in time might overlook the stable fingerprint and learn the temporary features instead. In general such an occurrence is known as *concept drift* – a change over time in the statistical properties of the class that the model is trying to predict. Therefore, the recognition might become less accurate over time. A model resilient against concept drift is the one that manages to capture the salient traffic features maximally correlated with the website fingerprint and thus remains performant over time. To reveal if our DNNs detect the actual website fingerprints and assess how well they perform in case of traffic changes, we train the models on a closed world and test them on data collected from visiting websites of the same closed world periodically over 2 months. In order to fairly compare DL-based methods to CUMUL, we have to evaluate them on the same dataset with the same number of traces. Due to CUMUL's scalability issue, the biggest dataset possible to use for this evaluation is CW_{200} with 2,000 training instances. Even though this is not the largest dataset we collected, it is still twice as large as the closed worlds normally used in prior works. Thus we train models on the whole CW_{200} dataset (with 2,000 training traces) and test them on the *revisit-over-time* dataset (as defined in Section 3.4).

The results are depicted in Figure 3.6 for DL and traditional CUMUL. The plot indicates the WF performance of various models trained on CW_{200} and evaluated on traffic re-collected 3 days, 10 days, 4 weeks, 6 weeks and 8 weeks after training.

The figure demonstrates how the classification accuracy decreases and the

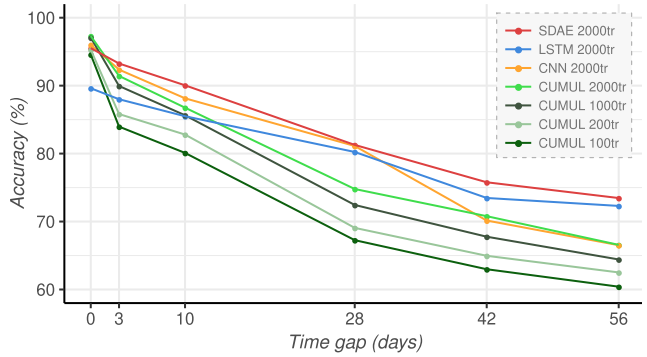


Figure 3.6: DL (SDAE, CNN, LSTM) vs. CUMUL resilience to concept drift: evaluation of CW_{200} over time.

classification loss increases gradually and drastically over time. These results illustrate the high generalizing abilities of both the evaluated models. Despite a significant 2-month time gap between the moment of training and the last evaluation, the DL algorithms are still capable to correctly deanonymize at least 66% out of 2,000 website visits. We witness a rather small accuracy drop in the first 3 and 10 days for all three DL models, which may be acceptable for an adversary who would prefer to use the built WF classifier for several more days rather than repeat the data collection and training process every day. In total, SDAE loses 22% of accuracy over 2 months, CNN loses 29%, while LSTM only loses 17%. Notably, being the most performant DL model on the day of training, CNN generalized worse than SDAE or LSTM. Despite the fact that the LSTM model (which still makes decision just based on the first 150 cells in the input sequence) is initially outperformed by both SDAE and CNN, after one month its accuracy catches up with that of the SDAE. Moreover, after 1 month the LSTM loss values are lower than those of the SDAE, which means that even though the LSTM outputs less correct predictions, it is overall more certain of these predictions. This obviously speaks in favor of LSTM's high generalization abilities, in line with our best expectations.

Our SDAE and CNN approaches outperform CUMUL with up to 7% over the course of 2 months. In total CUMUL loses 31%. LSTM network starts outperforming CUMUL after approximately 2 weeks. As such, this comparison not only shows that our approach indeed automates the feature engineering, but also that the learned implicit features (hidden in the neural network) are more robust against website changes over time. Notably, CUMUL is found to significantly improve its generalization abilities when trained on larger amounts of traffic traces per website, which proves that DL-based classifiers are not alone

in their requirement for a bigger training data for the highest performance.

The main conclusion here is that the DL-based classifiers are capable of extracting stable identifying information from the closed world traffic which allows for its deanonymization with a high success rate, even several days after training.

Open World Evaluation

This study compares DL-based WF attacks and CUMUL for the open world evaluation. The goal is to assess the classifier's ability to distinguish a traffic trace generated by a visit to one of the monitored websites from a traffic trace generated by a visit to any other unknown website. Our methodology for the open world evaluation differs from prior work in several aspects. We aim to provide a fair comparison of the classifiers by reducing possible bias. To this purpose we have to depart from the realistic WF setting and adapt the following assumptions:

- We model the monitored websites by training the classifier solely on the traffic traces of the websites an adversary is aiming to detect. By doing so, we assess the ability of the given learning algorithms to distinguish seen and unseen websites. Previous studies on WF argue that an adversary may improve the attack by additionally collecting and training on traffic of known websites that they are not interested in identifying, which is indeed a possibility given sufficient resources. But here we do not provide any helping patterns of the open Web to the classifiers to not distort their actual performance.
- We test the classifiers on balanced datasets: monitored and unknown websites in proportion 50%-50% (meaning that random classification would be accurate on average 50% of a time). Thus, we do not attempt to infer the realistic ratio, especially knowing that modeling an open world of a realistic scale poses large issues: (i) the effect of the hypothesis space complexity, as shown by Panchenko et al. [154], and (ii) the *base rate fallacy*, demonstrated by Juarez et al. [104]: even a highly accurate classifier trained on the monitored websites with a very low prior probabilities of visit cannot be fully confident of its predictions. Instead we assume a standard uniform probability distribution of visits to the monitored and unknown sets. With such evaluation the classifier's errors are more prominent and allow for a clearer comparison.
- Following the earlier reasoning, we use Alexa websites for both, monitored and unknown sets. Choosing a particular set of monitored websites

characterized by patterns that are not common to the whole Web would introduce classification bias with unpredictable impact on comparison. In order to objectively compare the studied classifiers, we demonstrate their abilities to distinguish seen and unseen fingerprints belonging to the websites of the same category (in our case, most popular websites).

We evaluate the open world WF attack for an adversary who monitors a set of 200 websites, while the target user may visit 400,000 more unknown websites. As a result, our open world dataset consists of 800,000 visits through Tor: one-time visits to 400,000 various websites in the Web and 400,000 visits to the monitored CW_{200}. We train the models solely on 2,000 instances of CW_{200} (thus obtaining the classifiers identical to those used for the closed world evaluation). Recall that earlier in the closed world evaluation section we already assessed their multinomial classification performance; the reported success rates indicate the ability of the classifiers to identify the exact visited monitored website. In this section we perform binary classification by testing the same models on our open world dataset. With this experiment we assess the classifiers' ability to recognize the input instance as a visit to a monitored or an unknown, earlier unseen website. The classifier makes decisions based on the cross-entropy loss function, which reflects its confidence in made predictions (Chapter 2 elaborates on the cross-entropy as a measure of classification confidence). If the loss value is low enough, the adversary assumes that the classified website visit belongs to a set of monitored websites. If the entropy is bigger than a certain *confidence threshold*, the adversary decides to not trust the classifier's class prediction and concludes that the tested traffic trace was generated by an unknown website, thus causing the prediction uncertainty. By varying the confidence threshold, the adversary balances the True Positive and False Positive Rate according to their priorities.

In our evaluation, we plot the ROC curve for the three DL classifiers in order to define the optimal confidence threshold which separates the monitored websites traffic from unknown websites traffic. Both CNN and SDAE again outperform CUMUL, if only slightly, as demonstrated by Area Under Curve values in the same figure. The ROC curves for SDAE, CNN and LSTM are depicted in Figure 3.7 and demonstrate the relative performance of the suggested open world WF DL-based attacks within 200 monitored and 400,000 unknown websites. We observe that the CNN model performs better than SDAE, and both perform significantly better than the LSTM model. However, the adversary may improve the models by using the open world traces for validation during hyperparameter tuning . LSTM classifier is outperformed by two other DL models because it only processes the first 150 Tor cells, opposed to 5,000 by SDAE and 3,000 by CNN.

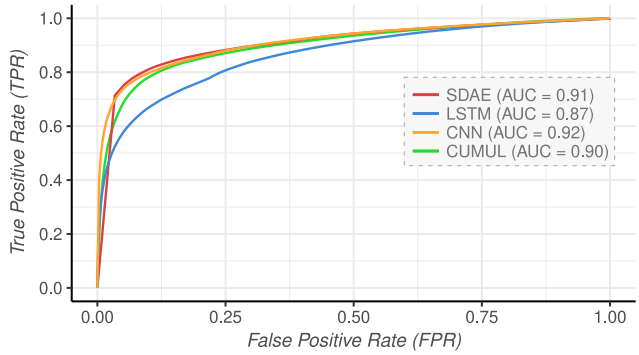


Figure 3.7: DL (SDAE, CNN and LSTM) vs. CUMUL in the open world setting for a monitored set of CW_{200}.

According to the ROC curves, an adversary may optimize the confidence threshold depending on their priority. For 200 classes, the categorical cross-entropy E varies between 0 (absolute confidence of the classifier's prediction) to 5.3 (absolute uncertainty). The optimization examples are given in Table 3.6, where reduced thresholds allow to decrease FPR.

Table 3.6: DL vs. CUMUL in the open world setting.

	Optimized for TPR			*Optimized for FPR*		
Model	E	**TPR**	**FPR**	E	**TPR**	**FPR**
SDAE	0.005	80.25 %	9.11 %	0.001	71.30 %	3.40 %
CNN	0.033	80.11 %	10.53 %	0.013	70.94 %	3.82 %
LSTM	0.062	76.19 %	19.78 %	0.010	53.39 %	3.67 %
CUMUL	0.048	78.00 %	9.89 %	0.018	62.57 %	3.58 %

Our open world evaluation considers a large set of unknown sites in which the adversary cannot train, allowing us to test the generalization of our models in a large sample of the Web. Similarly to the state-of-the-art, we observe how our DL-based approach withstands a challenging open world scenario, providing high accuracy on the largest set of unknown sites.

In the previous subsections, we have shown the relative performance of various DL models in comparison with each other and with the traditional CUMUL classifier. In certain experimental settings we improved beyond the state-of-the-art, e.g., in resilience to content changes and in success rate on the largest closed world. The success rates of WF attacks proved to depend on the

closed world size, the amount of training data available to the adversary and the computational resources that can be used to train the classifier. For the evaluations performed in this chapter, we used the resources available at our institution, but we acknowledge that a more powerful attacker could most likely further improve the attack by using more resources for data collection, model selection and training.

3.6 Discussion

In this section, we list limitations of this work and discuss remaining open challenges with regard to both the threat model and the deep learning methods presented.

As in virtually all prior work on WF, we analyzed the attacks only on visits to homepages and omitted other pages within the considered websites. We acknowledge this is an unrealistic assumption. However, as our main goal was to perform a fair comparison with existing attacks, we used the same experimental settings. As the models developed in prior work were tailored to these particular settings, the evaluation of techniques that consider inner web pages was deemed out of scope for this chapter. Nevertheless, we find automatic feature learning a promising approach to this problem.

We do not try to approximate the probability of visiting a closed world site vs. a site from the open world in our experiments. We assume that all open world sites have the same prior probability and all closed world sites have the same prior probability. We acknowledge this does not reflect reality but one can only hypothesize on the actual popularity distribution of websites over Tor without risking the privacy of Tor users. It is a limitation of our study and previous work.

Deep learning allows us to replace manual feature engineering with automatic feature learning. Therefore, the resulting attack is not defined by an explicit set of features that would be easily interpretable by a human analyst, but is instead based on abstract implicit non-interpretable features, being learnable parameters of the neural network. Moreover, these features have proven to be more robust to web content changes in comparison to those suggested in prior literature. Consequentially, the corresponding countermeasure cannot focus on concealing specific features as it was done earlier, but in order to defend against the DL-based attack we have to challenge the DL algorithm itself. Therefore, future work should focus on defending against the automated WF attacks, such as deep neural networks presented in this study.

One line of research for future work could be to investigate whether it is possible to mislead the deep neural network predictions. For instance, such research could base on the latest work on *adversarial examples* [34]. These are inputs to the learning model specifically crafted to fool the neural network into classifying them into a wrong class. Adversarial examples can be explored as a defense strategy against DL-based WF in order to protect Tor user's privacy.

In the very recent work by Wang and Goldberg [226], a defense technique based on half-duplex communication and burst molding is proposed. The authors assert that the novel defense defeats all WF attack techniques known to date. It would be useful to validate whether the claim would hold against automatic feature learning from a large volume of protected traffic.

3.7 Conclusion

In this study, we propose a new website fingerprinting attack based on deep learning. We assessed the attack feasibility through automated feature learning and showed that deep learning is capable of fingerprinting websites with an accuracy that is comparable to thee state-of-the-art techniques. The three deep neural networks we investigated have shown their strengths and weaknesses in the context of website fingerprinting: (i) SDAE performed well overall and proved to be the most stable DNN with respect to the closed world setting; (ii) CNN is the fastest network due to fewer learnable parameters, and performed best for smaller closed worlds and for the open world evaluation. However, this DNN has a higher risk of overfitting, which was revealed by the larger closed worlds and the concept drift experiments; (iii) LSTM performed the slowest, but exhibited the best generalization capabilities due to its recurrent structure. However, its constraint in backpropagation did not allow to process long traffic traces without jeopardizing the overall performance.

In certain experimental settings, our attack even improves existing implementations: (i) SDAE showed better results than CUMUL on the largest closed world we evaluated; (ii) all three DL approaches prove to be more robust against web content changes than CUMUL, with LSTM being twice more robust; (iii) SDAE and CNN networks perform slightly better in the open world evaluation than CUMUL; and (iv) the DL approach is generally more scalable due to parallelization and automated model selection. In conclusion, using deep learning gives an adversary major advantages, resulting in accurate and efficient traffic deanonymization.

Chapter 4

Traffic Correlation Analysis through Deep Learning

Preamble

Traffic analysis attacks against encrypted web traffic are a persistent problem. However, there is a large gap between the scientific estimate of attack threats and the real-world situation. As traffic analysis attacks depend on highly specific metadata information, they are sensitive to artificial changes in transmission characteristics. While the advent of deep learning greatly improves the performance rates of traffic analysis attacks on Tor in research settings, deep neural networks are known for being implicitly vulnerable to artifacts in data. Removing artifacts from our experimental setups is essential to minimizing the risk of evaluation bias. In this work, we study a state-of-the-art end-to-end traffic correlation attack on Tor and propose a novel data collection setup. Our design addresses the key constraint of prior work: instead of using a single proxy node for collecting exit traffic, we deploy multiple proxies. Our extensive analysis shows that in the multi-proxy design, (i) end-to-end round-trip times are more realistic than in the original design and that (ii) traffic correlation attack performance degrades significantly on realistic timings. For a reliable and informative evaluation, we develop a general scientific methodology for replicating and comparing machine- and deep-learning attacks on Tor. Our evaluation indicates high relevance of the multi-proxy data collection setup and the novel dataset.

This chapter is based entirely on our published study on methodologies for

data-driven traffic analysis on Tor [171]. Upon the publication of this work, we have open-sourced the data and the code to encourage further progress in the domain. Specifically, the data can be used for research on both website fingerprinting and end-to-end traffic correlation attacks against Tor. Having a single dataset for both these attack categories is important as we consider it likely that mitigations against one attack may positively or negatively impact the vulnerability of Tor to the other attack. This potential effect could, hence, be extensively analyzed by conducting a joint evaluation of both attacks on the same data with additional mitigations instead of isolated experiments.

This chapter generally illustrates that using powerful deep neural networks as side-channel attacks comes with several important caveats. First, if there are irrelevant artifacts in data, such as the biased timing measurements considered in this study, the network may silently learn from the artifacts and reach very high performance. In our work, we correct the data itself by isolating potential artifacts. However, in the absence of artifact-free data, deep learning attacks have to be evaluated more thoroughly, e.g., by integrating explainability analysis. This general topic is further explored in the context of defensive applications in Chapter 6. Second, the inherent high variability and non-determinism of deep learning demand more expansive performance measurements, where influential variables are controlled to get a more reliable view of the performance distribution. Comparing different attacks in the literature in terms of estimated performance distributions is necessary to arrive at generalizable conclusions.

4.1 Introduction

Encrypting traffic is a core security feature of the Internet. An additional layer of protection becomes available with the use of anonymity systems like Tor [161], where multiple layers of encryption and relayed connections conceal the activity of users. In this context, traffic analysis attacks on encrypted traffic are a persisting problem. They circumvent the data encryption by inferring sensitive information from the transmissions' *metadata*.

An important class of attacks in this context are *end-to-end traffic correlation* attacks. The adversary monitors traffic in both connection endpoints (client to entry guard and exit relay to server). Based on similarities between the incoming and the outgoing traffic at both ends, they can de-anonymize Tor connections [145, 146, 139, 52]. End-to-end attacks stand in contrast to *website fingerprinting* [154, 170, 193, 194, 24] considered in Chapter 3, where de-anonymization is achieved through classification of entry traffic. The current state of the art estimates considerably high success rates for both types of

attacks. However, ensuring *realistic* experimental setups remains challenging for academic work in this context.

Creating a realistic test setup is not trivial due to a diverse set of challenges. For example, as the critical evaluation by Juarez et al. shows [104], the world size for traffic analysis attacks still drastically limits the realism of existing performance benchmarks. The novel attack concepts and improved performance rates ought to be considered together with their experimental limitations.

For the evaluation of end-to-end traffic correlation attacks, the goal of a responsible measurement setup is particularly challenging. The adversary monitors traffic at *both* ends of the connection that makes use of a worldwide infrastructure. While it is trivial to gain access to the entry connection, there is no straightforward and privacy-preserving solution to accurately monitor traffic at generic exit links of the Tor infrastructure. Different workarounds allow us to circumvent this problem and still get access to the exit traffic. In the recent work by Nasr et al. [145], the use of a proxy node between the exit and target server provides access to the exit traffic. Despite the benefits of the proxy-based design, one proxy at a fixed location creates an *additional inter-continental hop* for a large fraction of connections but not for the others, and thus interferes with the *timing characteristics* of the dataset, which are often utilized in traffic analysis attacks [26, 86, 163, 24, 145]. The impact of such artifacts on the quality of data and, eventually, on the quality of attack performance estimations is unknown.

Another challenge of reliable empirical evaluations of traffic correlation attacks stems from the probabilistic nature of leveraged algorithms. Deep learning produces unprecedented performance in end-to-end side-channel attacks on time-series data, and various traffic analysis attacks against Tor are no exception [145, 170, 193, 24]. However, these high success rates come at the cost of increasing complexity and computational costs, performance instability and low interpretability. While exhaustive evaluations are infeasible, the progress of the field depends on our ability to draw reliable conclusions from limited empirical evidence. In particular, correct selection of metrics, proper tuning of complex models and fair comparison between them are all fundamental aspects.

In this work, we explore the impact of the characteristics of the underlying research dataset on effectiveness of end-to-end traffic correlation. Based on the preliminary statistical analysis, we introduce a novel multi-proxy data collection setup that overcomes the supposed key constraint of prior setups. In the second step of our study we replicate a state-of-the-art end-to-end traffic correlation attack DeepCorr [145] on our novel data. To this end, we design a replication methodology for data-driven attacks that utilizes appropriate performance metrics, accounts for various sources of bias and variability in machine and

deep learning evaluations and strives for reliable performance comparison. Our extensive experimentation shows consistently lower attack performance in the proposed multi-proxy setup, with high statistical significance. As a result, our work demonstrates a successful reduction of the timing bias introduced by the proxy usage.

In summary, this chapter makes three core contributions:

1. **Novel Multi-Proxy Setup.** We propose a novel data collection design for traffic correlation attacks in research settings that uses multiple proxies instead of a single proxy to capture exit traffic. Our strategic statistical analysis of Tor traffic transmission characteristics shows that the proposed design more closely approximates realistic timing measurements as compared to the state of the art.

2. **Attack Replication Methodology.** In order to accurately compare traffic correlation effectiveness in single- and multi-proxy data collection setups, we devise a general methodology for scientific replication and comparison of data-driven attacks on novel or modified data. The proposed algorithm aims to minimize evaluation biases and considers statistical significance of results, which improves over the common practices in the field.

3. **Evaluation Results.** Our empirical evaluation of an optimized DeepCorr attack demonstrates an average 7.95 % performance decrease in a more realistic multi-proxy setup. This outcome indicates that the proposed multi-proxy setup reduces the subtle bias introduced by the single-proxy design of prior work. We publish the corresponding dataset and code to facilitate further research.[1]

4.2 Technical Background

In the following, we document the technical background of Tor, provide a general overview of traffic analysis attacks, and discuss existing attacker models.

4.2.1 Circuits and Connections

Tor circuits usually consist of three relays that transmit information between the client and the server. As such circuits are part of the application-layer

[1] `https://tor-tc-dl.distrinet-research.be/`

security of Tor, they do not directly transport TCP packets, but handle multiple TCP streams of, e. g., a website request. On startup and during runtime, Tor continuously creates new circuits and discards older ones depending on different features, e. g., their lifetime or status flags. A standard circuit consists of an entry guard that connects to the client, a middle relay as the intermediate connector, and the exit relay that connects to the requested destination. The entry guard is picked from the user's guard set, which consists of a static selection of candidate nodes and gets updated after several months [85]. All remaining nodes are selected randomly from the consensus following a bandwidth-oriented preference.

A website visit results in multiple TCP streams that load the (sub)resources of the site. For a new website visit, Tor checks all currently available circuits and determines the one that best suits the current request. It then attaches all streams of that request to the circuit. After a circuit has been used for a connection, it is marked dirty and will not be reused.

4.2.2 Traffic Analysis Attacks

Traffic analysis attacks enable an adversary to learn sensitive information from the encrypted transmissions of a network. In the context of Tor, the worst-case leak of sensitive information allows the adversary to de-anonymize the connection.

Two primary types of traffic analysis attacks are Website Fingerprinting (WF) and End to End (E2E) traffic correlation, with the latter being the focus of this study. E2E correlation attacks focus on the similarities of transmissions at two endpoints of the connection, i. e., at the *entry* and the *exit* relay of the Tor circuit. In this setting, the entry connection reveals the IP address of the victim and the exit connection reveals the IP address of the server. The de-anonymization is successful if the adversary can find a related pair of entry and exit traffic. The success of an E2E attack is mainly influenced by the access to traffic in the network.

4.3 Challenges of Data Collection

Accessing Tor exit traffic to evaluate traffic analysis attacks in practice requires the use of a proxy in the connection between the exit relay and the server. Although it is technically possible to achieve this by running an exit node, this poses various ethical concerns (collecting data of other Tor users) and

other restrictions (lack of intrinsic traffic patterns of real-world Tor exit nodes). When using a proxy setup, an extra hop is added to the connection, which affects timing characteristics of the monitored traffic. So far it has not been investigated what the consequences of these alterations for the attack and its performance are. Prior data-driven state-of-the-art E2E attacks [145] used a single proxy server located in the US. While the authors document that the additional latency leads to performance impairments *for the attack*, we run a series of experiments to empirically analyze the impact of an additional inter-continental hop in different proxy setups in practice. We argue that a single proxy introduces different amounts of timing overhead, e. g., depending on how many inter-continental hops are part of the end-to-end route. Such timing differences might affect the distinguishability of traces from different groups of circuits. We further analyze how realistic the resulting overall timings are, given that a real attacker would not use a proxy at all but instead extract the traces directly from the exit connection.

4.3.1 Measurement Setup and Overview

Over a period of one week, we run a client in our institution to measure Round-Trip Times (RTTs) to each of the top 100 websites in a recent Tranco list[2] [114] through 15,935 standard three-hop Tor circuits. In addition, we collect RTTs to 14 geographically distributed servers, located in the most important data center locations in the world (cf. Figure 4.1). Finally, we measure the RTTs from each of these servers to each website. To measure the RTT to a website, we send an HTTP `HEAD` request and measure the time until our client receives the response. To be able to compare these timings to RTTs between our client and the servers, we run a simple HTTP response server on each server that also responds to HTTP `HEAD` requests by our client.

For each proxy, we combine the time from our client to that proxy with the time from the respective proxy to each website. This way, we are able to simulate the timing overhead introduced by different proxy setups and compare different proxy setups with each other.

4.3.2 Analysis of RTT Measurements

We now present the results of our preliminary timing measurements. In particular, we conduct the following analyses to evaluate the practical impact of different proxy setups on timing characteristics of Tor traffic:

[2] `https://tranco-list.eu/list/4L8X`

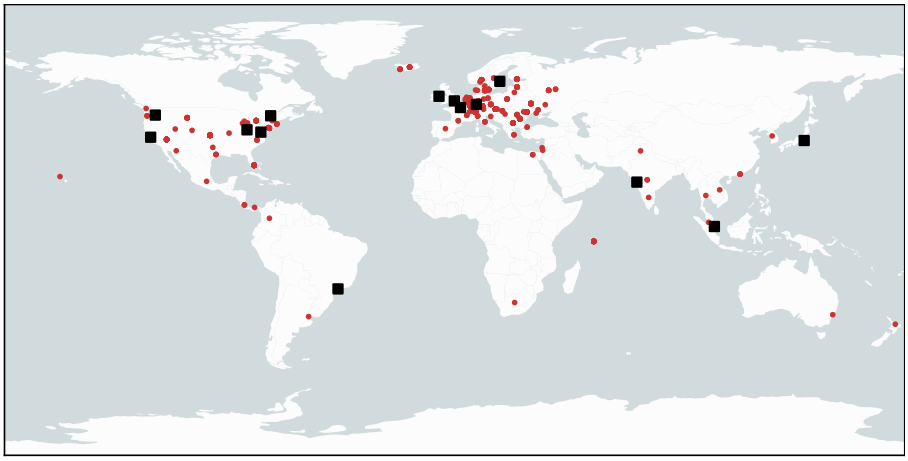


Figure 4.1: Overview of exit relay locations (red) and locations of proxies (black) in the preliminary analysis.

1. We analyze differences in timings from our client to a single proxy in the U.S. between groups of circuits, that are defined by the number of inter-continental transmissions between two subsequent hops in the connection.

2. We compare timings measured with two proxy setups, i. e., a single proxy in the U.S. and 14 geographically diverse proxies to the timings a realistic adversary without a proxy would see.

3. We quantify the overhead introduced in the timings depending on the number of proxies in our set of 14 proxies.

Inter-Continental Connections in DeepCorr

We argue that relaying all Tor traffic through a single proxy to evaluate the attack affects the end-to-end RTTs of monitored Tor traffic. This may lead to unknown consequences for other timing features that may affect the distinguishablity of traces within the feature learning process. For Tor circuits whose exit is located in Europe, the use of a single proxy in the U.S., as done in state-of-the-art traffic correlation attack evaluations [145] introduces the timing overhead of an inter-continental hop at the end of the connection.

We generalize this assumption and compare the end-to-end timings between our client and websites accross groups of circuits that are defined by the number

Table 4.1: Number of inter-continental connections within Tor circuits. Since our client is located in Europe, the number of hops is always uneven in the single (US) proxy setup. Only circuits with all relays in Europe or North America (i. e., 96 % of all circuits) are included.

Hops	Single Proxy	Multi-Proxy
0	–	8,653 (54.3 %)
1	10,811 (67.8 %)	2,158 (13.5 %)
2	–	4,131 (25.9 %)
3	4,468 (28.0 %)	337 (2.1 %)

of inter-continental hops in the connection. For the sake of simplicity, we only consider circuits with all relays located in Europe or North America, i. e., 96 % of all circuits in our set of 15,935 standard 3-hop circuits. Table 4.1 lists all circuits in our set depending on the number of inter-continental hops in the connection for both the single US-proxy and multi-proxy setup. Due to our client being located in Europe, the connection to the US-based proxy includes either 1 or 3 inter-continental hops. Effectively, the numbers of circuits with 0 and 2 inter-continental hops in the multi-proxy setup constitute those circuits, that include an inter-continental hop between exit relay and US proxy in the single proxy setup. That is, for 80 % of circuits, the single proxy setup introduces an *additional* transfer between Europe and North America.

Introducing an additional hop in the connection may be a critical issue since it affects round-trip times between client and proxy with potential consequences for the feasibility of traffic analysis. Figure 4.2 illustrates the distributions of RTTs between client and proxy depending on the number of inter-continental hops in the single proxy setup. We also distinguish whether all hops occur within the Tor circuit (*Tor-only*) or if one hop is introduced by the *Proxy*. Whereas we see differences in timings depending on the number of inter-continental hops (1: $mean = 0.68\,s$, $sd = 0.42\,s$, 3: $mean = 1.01\,s$, $sd = 0.36\,s$), it does not seem to affect the timings where the hop occurs, whether it is in the Tor circuit or between exit and proxy.

Single Proxy vs. Multi-Proxy vs. No Proxy

In the next step, we compare the RTTs to all websites in our dataset through all Tor circuits in our dataset. We use three different proxy setups, (i) a single proxy server located in the US (East), reflecting the setup used for state-of-the-art evaluations of traffic correlation [145], (ii) a set of 14 proxies out of which we select the one that is located nearest to the exit for each circuit, and

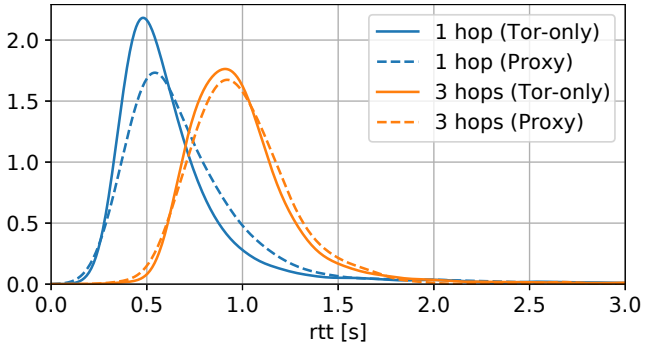


Figure 4.2: Distribution of RTT between client and proxy by the number of inter-continental hops in the connection. Tor-only denotes that all hops occur within the Tor circuit, proxy denotes that the last hop is between exit relay and proxy.

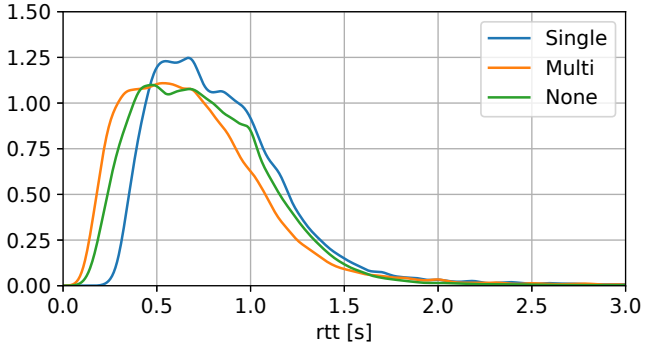


Figure 4.3: Empirical distributions of end-to-end RTTs between clients and websites in three proxy setups.

(iii) no proxy, which reflects timing characteristics of traffic a real adversary would access directly in-transit. For the sake of simplicity, we technically do not measure end-to-end RTTs but collect (i) timings through each Tor circuit to each proxy, (ii) timings through each Tor circuit to each website without a proxy, and (iii) timings from each proxy to each website. We then effectively simulate the different proxy setups, by composing respective timing segments to end-to-end RTTs.

Figure 4.3 shows the distributions (probability density functions) of end-to-end RTTs in each of these setups. We see that the timings in the *multi*-proxy setup ($mean = 0.72\,s$, $sd = 0.43\,s$) are more similar to the setup without proxy (*none*, $mean = 0.76\,s$, $sd = 0.38\,s$) than those in the *single*-proxy setup ($mean = 0.86\,s$, $sd = 0.41\,s$). The overlap of *single* and *none* distributions is 77 %, whereas it

is 84 % for *multi* and *none*. We also notice that timings in the multi-proxy setups tend to be even *lower* than in the setup without a proxy. We attribute this to our proxies being located in large data centers and presumably better connected to many websites (also located in data centers) than Tor exits are. Whereas the multi-proxy setup also affects the RTTs (we discuss this limitation in Section 4.7), the effect is yet smaller in absolute numbers than for the single-proxy setup. Therefore, we argue that the multi-proxy setup is a better approximation of a setup without a proxy w. r. t. their timing characteristics, which is relevant for traffic correlation attacks.

Differences between timing distributions even increase when we compare the timings measured for each website. Table 4.2 lists the average RTTs for the three proxy setups and the timing distribution overlap for each of the single and multi proxy with the no proxy setup. For most websites, the distributions of measured timings in the setup with multiple proxies are more similar (i. e., higher overlap) to the setup without proxy, than the timing distributions of the single proxy setup.

Table 4.2: Mean end-to-end RTTs and timing distribution overlap.

Website	**Multi** [s]	**Overlap** (M,N)	**None** [s]	**Overlap** (N,S)	**Single** [s]
`google.com`	0.68	71 %	0.77	51 %	0.95
`youtube.com`	0.68	70 %	0.78	50 %	0.95
`facebook.com`	0.75	91 %	0.76	41 %	1.23
`netflix.com`	0.69	79 %	0.76	58 %	0.90
`microsoft.com`	0.98	88 %	1.00	41 %	1.23
`twitter.com`	0.74	86 %	0.77	85 %	0.81
`instagram.com`	0.66	82 %	0.71	39 %	0.94
`apple.com`	0.62	76 %	0.71	77 %	0.76
`linkedin.com`	0.62	82 %	0.67	75 %	0.76
`qq.com`	1.03	72 %	1.12	75 %	1.22
all (top 100)	0.71	84 %	0.75	77 %	0.85

Number of Proxies

Our next goal is to estimate the required number of proxies to limit the transmission overhead. Whereas we have seen that selecting the nearest one out of a set of 14 server instances better reflects the timing characteristics of Tor traffic without an exit proxy, we now investigate to what extent the transmission overhead depends on the number of proxies to choose from. For each circuit

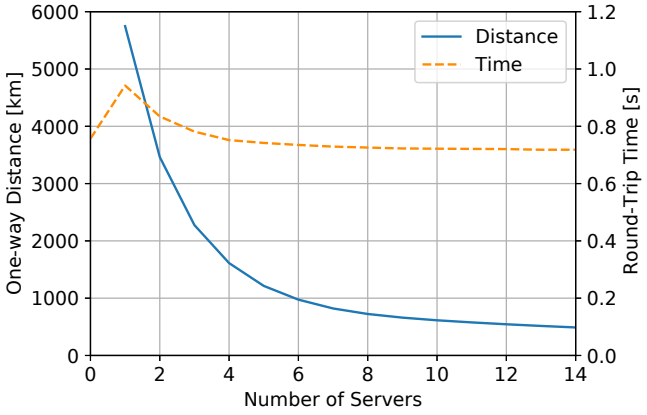


Figure 4.4: Distribution of Tor's infrastructure by fractional numbers of nodes and bandwidth shares per country.

in our dataset, we generate sets of n randomly selected proxies (for each n from 1 to 14) and calculate the distances from the exit relay to the nearest proxy in each set. Our results in Figure 4.4 show that the average minimum distance of more than 5,700 km when only one proxy server is used, is reduced to approximately 1,000 km when 5 servers are used, and to less than 500 km with 14 server instances in place.

In addition, we also measured the timing overhead per number of server instances used. Our results (cf. Figure 4.4) show that, while the average RTT is around 0.75 s without a proxy, we measure an average RTT of 0.94 s in a single-proxy setup, effectively introducing 190 ms overhead. When we increase the number of proxies to up to 14, the average RTT gradually approximates the value of the setup without a proxy. In summary, our observations suggest that using a single proxy in the U.S. for monitoring Tor exit traffic introduces a specific timing overhead and, therefore, might also affect timing features that play a role in the evaluation of end-to-end confirmation attacks. We assume that a setup incorporating multiple proxies closer to the exits is a better fit for simulating timing characteristics and, therefore, traffic a realistic adversary would monitor in practice without a proxy.

4.4 Data Collection Setup

In the following, we document the experimental setup that we use to record traffic for evaluating end-to-end confirmation attacks.

4.4.1 Requirements

The setting of E2E attacks requires recording traffic at both endpoints of a connection. In the following, we discuss the implications of designing the experimental setup for an E2E adversary. In general, the dataset can also serve for evaluations of WF attacks (both closed- and open-world) that are conducted with entry traffic only, which is a subset of the data we collect.

Proxy Setup

A recent attack by Nasr et al. [145] uses deep learning for an E2E attack. In their experimental setup, the authors use a `SOCKS5` proxy between the exit relay and server of a connection to be able to record traffic at both ends of the live Tor network. Using an additional proxy has the disadvantage of adding another hop to the connection, which extends the transmission distance and changes the direct connection between the exit relay and the server. However, in contrast to using a dedicated exit relay, it does not introduce any ethical complications and can be adjusted in a way that the additional overhead can be minimized. The following characteristics must be considered when using an additional proxy.

Circuit Diversity. During one browsing session, Tor creates multiple circuits that are used and switched on a regular basis. Despite a primary entry guard that remains the same as long as possible, the remaining middle and exit relays change for different circuits. Because of Tor's worldwide distribution of relays, each circuit has individual performance characteristics such as the circuit length or the use of advertised bandwidth. Using a proxy allows maintaining the original circuit diversity.

Server Locations. Both attacks require the generation of a representative dataset including website requests to a high number of different server locations. Along with the distribution of exit relays, this creates a high diversity in the last hop of the connection. Examples of this are varying connections' lengths between the exit and server and individual transmission characteristics, respectively. Using multiple proxies at different strategic locations allows to limit the overhead of the additional hop (cf. Section 4.3.2).

Onion Service Traffic. Using the `Alt-Svc` header, web services can indicate that the website can be accessed over an alternative service. For instance, when the website detects that it is being accessed over a Tor connection, it can refer

the user to the relevant `.onion` address. The user agent will then initiate a connection to the onion service and use it to request all subsequent resources. Because of our proxy-based setup, websites receive connections from a cloud provider and not a Tor exit node. Furthermore, as all client traffic is direct to the proxy, no connections to onion services will be made. In a real-world scenario, traffic to the onion services would still appear in the compromised guard node but not in the exit node. Consequently, this minimally affects website fingerprinting attacks, but needs to be considered by the adversary in an E2E confirmation attack. We further discuss this limitation in Section 4.7.

4.4.2 Setup Overview

The proxy setup consists of a series of clients, a central management entity, and an additional proxy between the exit and server of a connection (cf. Figure 4.5). The clients run in virtual machines in our institute's cloud infrastructure and consist of the Tor Browser Bundle (TBB), `3proxy`, which allows proxy chaining in addition to Tor's `SOCKS5` proxy, and the core Tor implementation that establishes circuits and uses them to connect to web servers. At each end of the connection, we use `tcpdump` to capture the entry traffic between the client and the entry guard and the exit traffic between the exit relay and the server. The central management scripts organize the website crawling and orchestrate `tcpdump` at both endpoints of the connection.

Proxy Chain

`3proxy`[3] is a lightweight proxy client and server that allows us to put an additional proxy between the TBB and core Tor implementation that connects to the corresponding proxy between the exit and the server. As one goal is to minimize the overhead of the additional proxy hop, we run multiple entities at strategic positions and pick the location with the minimum distance between exit and proxy for each circuit established. To this end, we run 18 proxies in 6 different locations (1 in London, 1 in San Francisco, 3 in New York, 2 in Amsterdam, 1 in Singapore, and 10 in Frankfurt). These proxy locations reflect the distribution of large data center locations around the world. Each proxy has an individual port assigned on the client side. Depending on the circuit that will be used next, we pick the best fitting proxy by deciding for the port that matches the correct server entity. At this point, it is very important to select a proxy before the stream, i. e., the TCP connection, is attached to circuit.

[3] `3proxy.ru`

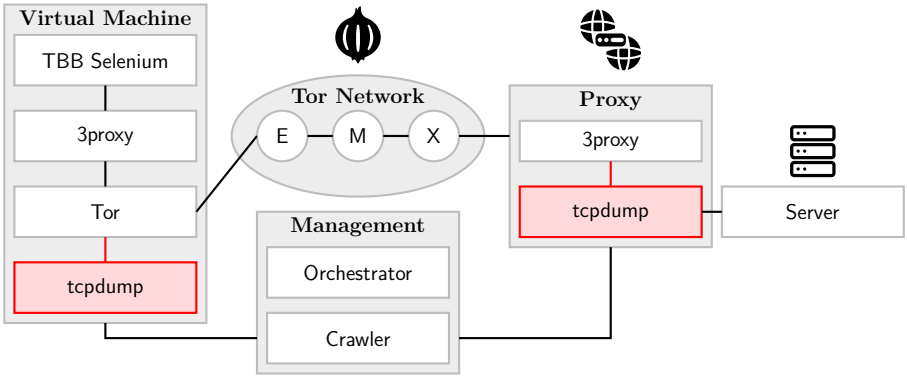


Figure 4.5: Experimental setup for data collection. Tor nodes: entry (E), middle (M) and exit (X).

Therefore, we must predict the circuit that will be used next and attach it to the fitting port and proxy combination.

Circuit Prediction

If we do not want to interfere with the circuit establishment procedure and selection of relays, we must pick a proxy according to the circuit used in a connection. This is challenging, as Tor builds circuits in advance, and only decides for one of them at the moment the client initiates a connection to a server. However, using an additional proxy between the exit and the server requires using a chain of proxies that adds our own `SOCKS5` node behind the initial proxy that connects the Tor client to the entry relay. Technically, we must set up this proxy chain at a specific port that later is used by Tor for the connection establishment. That said, we need to pick the correct candidate from the set of available circuits that Tor will use for the next connection, identify the best (closest) proxy for the exit of this circuit, set up the proxy chain at the port Tor will use, and then wait for the connection to be made.

To pick the correct circuit, we implement a small patch in the Tor client that copies the original circuit selection procedure from Tor, i. e., we apply the same rules as Tor to determine the upcoming circuit. The patch is a control port implementation that iterates over all existing circuits and returns the one that is most likely to be picked for the next connection. We use this information for two purposes. First, we estimate the location of the exit relay in the circuit to prepare the *closest* proxy for this connection. Second, we use the IP of the exit relay to allow the incoming connection at the proxy. This allows us to

restrict any other connection attempt that is unrelated to our experiments. A side effect of these access rules is that only correctly picked circuits succeed. In case we pick the wrong circuit (and consequently a false proxy location), the incoming connection uses a different IP address and is blocked at the proxy.

4.4.3 Data Collection Details

In total, we collected three datasets that are used in the experiments presented in the remainder of this chapter. Each dataset was collected on a distributed crawler setup using the latest Tor Browser version (9.0.1) which was orchestrated through tbselenium [2]; for the SOCKS proxy, we used 3proxy (version 0.9). For each data collection, the most recently available Tor version was used, with the extension that allows us to predict the location of the exit node. The 64 instances of the crawler were distributed over 8 virtual machines that were each provisioned with 4 vCPUs and 8 GB of RAM. The first two datasets, *SD* and *MD* consist of visits to the 15,000 most popular websites according to the Tranco ranking[4] [114]. For the *SD* dataset, only a single proxy run on Amazon's Northern Virginia (us-east-1) datacenter was used. For the *MD* dataset, we leveraged proxies that were globally distributed over six Amazon datacenters, covering America, Europe and Asia. The data for these two datasets was collected between July 28, 2021 and August 20, 2021. In the final largest dataset, MD_{1mln}, we collected traces for the Tranco top 1 million sites[5], also using proxy servers geographically distributed over six locations. The data collection for this dataset started on March 5, 2020 for a duration of 8 days. All datasets will be made publicly available in a parsed form; the raw PCAPs are available upon request.

4.5 Evaluation Methodology

In this section, we outline our evaluation methodology. The main goal is to establish whether the choice of a data collection setup indeed impacts the estimated feasibility of traffic correlation attacks. Previously we have shown that the multi-proxy setup is much better at approximating the timing characteristics of traffic in the real Tor infrastructure. However, it is still unclear whether the disturbed timing measurements in the single-proxy setup make a simulated traffic correlation attack easier to deploy. With this evaluation, we aim to empirically compare performance of the state-of-the-art data-driven E2E attack

[4] `https://tranco-list.eu/list/65VX`
[5] `https://tranco-list.eu/list/KW8W`

(DeepCorr[145]) on two different datasets: *SD* collected through a single-proxy setup from prior work, and *MD* collected through a multi-proxy setup proposed in our study.

4.5.1 Validity

Comparing two datasets is not a trivial task due to a number of influential factors that are largely out of control of researchers. Such factors, like the current state of the Tor infrastructure, the amount of users and the dynamically changing web content, may impact the measurements in an unpredictable manner. We call these factors *the impact of time*. In order to minimize the impact of time on our two datasets and thus provide a consistent ground for comparison, we have already taken the following steps in Section 4.4:

1. Both datasets are collected within the same time frame: August 2021.
2. We fix the Tor process version to 0.4.6.6.
3. We use the same set of 15,000 websites for one-time visits in both datasets. After filtering the failed web visits and traces that may be too short for learning (less than 100 packets), we fix a set of 7948 websites that are present in both datasets.
4. The only controlled variable is the amount of proxies used in the setup: one for the single-proxy evaluation and 18 for the multi-proxy evaluation.

With this set of actions, we try to ensure that any differences in data originate from the number and locations of proxies, and not from artifacts or inconsistencies in data collection.

Furthermore, for a valid comparison between two datasets we recognize several influential aspects of developing and evaluating data-driven attacks. These include (i) correctly chosen performance metrics, (ii) a proper attack replication algorithm for evaluating deep learning models on novel data, and (iii) explainability analysis. Careful incorporation of these aspects will ensure that the observed difference in attack performance can be attributed to the actually present semantic differences in data, and not to incorrect interpretation of results, biases or chance. This is especially important for attacks based on deep neural networks, whose learning mechanisms are hardly interpretable and heavily rely on randomness and a multitude of hyperparameters with complex relations. Further in this section, we explain each of these parts of our evaluation methodology.

4.5.2 Performance Metrics

Scientific empirical evaluations of traffic analysis attacks against Tor aim to estimate the technical threat to privacy of Tor users. However, in lab conditions, these attacks are deployed in simulated scenarios that operate under a set of simplified assumptions, which makes performance estimation technically feasible and systematic, but inevitably limits the attack realism. E.g., related work has focused on perfecting performance estimation for WF attacks, to respect the open-world evaluation challenges [104, 222]. Estimation of actual E2E traffic correlation attack performance for powerful adversaries in the wild is extremely difficult in lab conditions, as it depends on the realism of the threat model in terms of coverage of Tor relays and amount of concurrent connections. However, in this chapter, we set the scope on the realism of measurements for E2E attacks affected by *the choice of the data collection setup* as is used in scientific studies. For comparing relative attack performance on similar data with two different data collection setups, a fixed large number of traces is sufficient. Nevertheless, in a simplified research setting the exact choice of metrics and correct interpretation of results are especially important. We look into metrics used by the state-of-the-art and propose improvements.

In prior work, a common approach employed to estimate E2E correlation performance is to report and compare the False Positive Rate (FPR), exemplifying the fraction of non-associated pairs that were incorrectly predicted as correlated, and the True Positive Rate (TPR), or Recall, that illustrates the fraction of truly associated pairs that were rightfully predicted as correlated. We argue that using these scores for performance estimation is insufficient and may even be misleading, as they do not provide a complete view over the attacker's capabilities. The reason is the *base rate fallacy* specific to the E2E attack use case. This problem originates from the *inherent imbalance* of positive and negative pairs in the test set: the adversary has to correlate all combinations of traces ($N \times N$), while only 1 out of N intercepted connections is truly associated. In other words, the base rate of positive samples is very low – $\frac{1}{N}$, and decreases with the growth of the volume of traffic. With such a low base rate, even a high TPR and a seemingly low FPR do not make the attack reliable.

To understand this with an example, consider the asserted optimal performance values of DeepCorr: a TPR/Recall of 0.80 and a FPR of 10^{-3} obtained for a number of associated pairs $N = 25,000$. While this may seem effective, when translated to absolute numbers, the actual True Positives amount to $TP = TPR \times \#pos = 0.80 \times 25,000 = 20,000$, while the actual False Positives amount to $FP = FPR \times \#neg = 0.001 \times (25,000 \times 24,999) = 624,975$. This means that for each correctly detected associated pair of flows, the DeepCorr

model mistakenly predicts with the same confidence around 30 non-associated pairs as correlated, and is therefore extremely *imprecise*. This issue is not visible from TPR and FPR, or the corresponding ROC-curve, alone. What happens is that precision is ignored during optimization: for this model, precision in fact equals $Pr = TP/(TP + FP) = 20,000/(20,000 + 624,975) = 0.031$, with the maximum possible value of 1.0 and a baseline of $25,000/(625 \times 10^6) = 0.00004$.

DeepCorr performance is decidedly superior to the E2E correlation attacks previously reported in the literature [146, 139, 52, 131], and further improves when using more packets in a flow. However, it is crucial to report all relevant metrics for the sake of transparent and informative estimations. Depending on the attacker goals and the costs of false positives and false negatives, one may choose to optimize the *precision-recall trade-off* to obtain a more trustworthy (precise) detector.

Prior work optimizes the attack based on *loss* – the optimization function used in deep neural networks to learn from data and to detect when to stop learning. However, specifically for DeepCorr, there is no theoretical guarantee that the optimal loss will correspond to optimal precision-recall trade-off (the final goal of the attack). So instead, an adversary may choose to also evaluate a chosen domain-specific metric on every training epoch in addition to the standard loss. In our evaluation, we tackle the inherent disconnect between loss and attack target metrics by optimizing for Average Precision (AP) that summarizes area under the PR-curve.

4.5.3 Attack Replication

We aim to analyze the performance difference of E2E correlation attack when deployed on a dataset collected with a single proxy versus a dataset collected with multiple proxies. Higher attack performance on the single-proxy dataset would confirm that the deviation of end-to-end timing characteristics from the real-world setup favorably impacts attack simulation. At the same time, lower attack performance on the multi-proxy dataset would indicate that correlation of traffic is a harder problem than it appears from prior research, and including multiple proxies in data collection is a more accurate way to model the problem. The difference in E2E attack performance thus serves as *an indirect signal of quality of the two data collection setups* and is obtained through experimental analysis.

The challenge of experimental analysis is in deriving reliable and generalizable conclusions from limited empirical evidence, which is especially pronounced with data-driven approaches. WF and E2E correlation attacks have long leveraged traditional statistical methods and machine learning algorithms, before at last

considering deep learning in 2018 as a novel, powerful approach for both Tor traffic analysis attacks [170, 145]. While relaxing the need for domain knowledge through eliminating manual engineering of salient traffic features, deep learning does increase the complexity and is overall not straightforward in deployment. A common challenge comes from the need to replicate state-of-the-art deep learning attacks on a novel dataset collected in different conditions. In this case, it is not sufficient to reuse the exactly same configuration of a deep neural network reported in prior work. The reason is that the change of any origin in the underlying data distribution may affect the nature and difficulty of the learning problem. To account for that, the deep learning attack needs to be retrained on novel data.

In the context of Tor, many factors may affect the underlying data distribution in traffic and thus change problem complexity. These could be fundamental updates in the protocol itself, such as defenses against traffic analysis. A deep neural network tuned on unprotected traffic is unlikely to possess enough learning capacity to learn the problem on a harder concept of defended traffic and thus needs to be retuned and then retrained. Moreover, there could be changes in the input representation, such as addition or omission of information or input preprocessing. Finally, as is the case in our study, the impact of time and changes in the data collection methodology may affect the data distribution and the overall problem complexity.

Therefore, to properly deploy and compare DeepCorr on our two new datasets, we develop an *attack replication algorithm* for evaluating and comparing a deep learning or machine learning-based attack *on different datasets*. The general replication algorithm is depicted in Algorithm 4.1, it is applicable to all the aforementioned cases and trivially extends to more than two datasets. Further on we clarify some of the steps of the algorithm and discuss why they are meant to account for the primary sources of evaluation bias.

Hyperparameter Tuning. There are several important considerations regarding tuning. Firstly, while the attack used for replication (e.g., a state-of-the-art DL approach) is the same for both datasets, it has to be re-tuned for each given dataset separately for fair comparison. This is to ensure that each dataset is assessed with its own best model configuration (architecture and hyperparameters). Not only because that would be the procedure followed by a real-world adversary, but also because with the best configuration, we are able to more accurately assess the expressive power of the dataset.

Secondly, we need to consider resource constraints, as the algorithm involves many intensive iterations through the datasets. There is a trade-off between on one hand a thorough evaluation that addresses possible biases, and on the

Algorithm 4.1: Replication and comparison of a ML/DL attack on two different datasets. Here, with AP as a target metric and t-test for statistical significance.

Input: Datasets D_1 and D_2, ML/DL attack A, hyperparameter search space S.
Output: Dataset with highest mean attack performance; the difference between means; statistical significance.

1 Set number of cross-validation folds K
2 Set maximum number of attack configurations C
3 Set number of randomized training runs n
4 $AP_{D_1} \leftarrow \{\}$, $AP_{D_2} \leftarrow \{\}$
5 **for** *dataset* $d \in \{D_1, D_2\}$ **do**
6 Split d into K equal folds $d_{1,...,K}$
7 **for** $k \in \{1, ..., K\}$ **do**
8 Put aside fold d_k as $test_k$
9 Assign the remaining folds to $train_k$
10 Sample $tune_k \subseteq train_k$
11 $AP_{best} \leftarrow 0$
12 **for** $i \in \{1, ..., C\}$ **do**
13 Sample config c_i from S using the chosen optimization strategy
14 Split $tune_k$ into $tune_{train}$, $tune_{val}$
15 Train A with config c_i on $tune_{train}$
16 Compute AP on $tune_{val}$
17 **if** $AP > AP_{best}$ **then**
18 $c_k = c_i$, $AP_{best} = AP$
19 **repeat** n **times**
20 Change random seed
21 Train A with config c_k on $train_k$
22 Compute AP on $test_k$
23 $AP_d \leftarrow AP_d \cup AP$
24 Compute t-test and p-value on (AP_{D_1}, AP_{D_2})
25 **return** *argmax*$(\overline{AP}_{D_1}, \overline{AP}_{D_2})$; $|\overline{AP}_{D_1} - \overline{AP}_{D_2}|$; *p-value.*

other hand staying within reasonable computational time. In terms of tuning, the amount of required resources is mainly controlled by the size of a search space S, the maximum number of evaluated attack configurations C, and the chosen size of a subset sampled from the training data for tuning (step 10), which could be a smaller fraction or a full training set.

Nevertheless, it is important to use a consistent optimization strategy that determines which hyperparameter configuration is evaluated at each iteration (step 13). The common options are Grid search (exhaustive search), Random search and Bayesian optimization (probabilistic search). We use Bayesian optimization and elect to start from a small model and gradually increase its

complexity. This approach is consistent with the heuristic in Bayesian statistics of preferring simpler models when performance is similar, as they are more likely to represent the underlying distribution of data more accurately [102].

Cross-Validation. To counter selection bias which could arise from evaluation on one fixed data split, the algorithm deploys a procedure similar to the classical k-fold cross-validation: different portions of the dataset are used to train, validate and test the model on different iterations [141]. This way we can assess if the results generalize well, i.e., the attack performs better on one of the datasets over most or even all of the folds. This analysis may increase the confidence in the outcome, or on opposite may show that the result is inconclusive, when the behavior is not consistent across the K folds.

Randomization. We also account for the non-deterministic nature of deep neural networks. Namely, we do several runs (n) for each cross-validation fold with a varied random seed, which impacts random initialization of weights in a deep neural network and random data shuffling between training epochs. Experimenting with more random seeds allows to assess stability of the attack and solidifies the observed trends in performance.

Statistical Significance. The algorithm returns the dataset that yielded the highest average attack performance based on the chosen metric, and the difference between the two averages. It is crucial to compute statistical significance to assess generalization and reliability of the findings, instead of reporting best runs or averages alone, which could be random and hence uninformative or misleading. Our algorithm assesses significance by performing a two-sample t-test, which is applicable to comparing performance of data-driven models with equal variances [56].

For completeness, in Appendix B.3 we provide Algorithm B.1 – a version of Algorithm 4.1 that compares *two different data-driven attacks on the same dataset*, to diminish evaluation biases when proposing new attacks.

4.5.4 Explainability Analysis

Our data collection approach and attack replication algorithm are meant to ensure validity of the conducted comparison between the single- and multi-proxy setups. The evaluation aims to assert one of the two datasets as the harder one to attack, thus indicating a quality difference between the two data collection setups. However, drawing strong claims from numerical performance metrics

is challenging, due to the high complexity and limited interpretability of deep neural networks.

In order to further support the results of the evaluation, we conduct additional explainability analysis of the two attack models. We aim to assess whether the models trained on different datasets utilize the inputs differently, e.g., by prioritizing different features.

We conduct explainability analysis through *visualization of activation maps* – a known technique in computer vision that is most commonly applied to Convolutional Neural Networks (CNNs) for image classification [238]. This approach allows to directly analyze the importance of input features for prediction, because it is applied in the first layer of the network, where the input is still interpretable.

Specifically, we look at the output of the first convolutional layer of the network, which produces a volume of activation maps – two-dimensional arrays of activations. These activations are the inner products of the chunk of the input trace and the convolutional filter (kernel). Because the size of one activation map is linear to the size of the input trace, we can relate the highest activation values to certain areas in the original input. Such areas are considered important by the kernels in the first layer. By repeating this process across all activations maps in the first layer and averaging across many input traces, we reveal which areas in the traffic trace the model finds most relevant for correlation. In our experiments we perform this analysis separately for truly associated and non-associated pairs for each dataset and report the results.

4.6 Experimental Results

In this section, we apply our proposed methodology aiming to compare performance of the state-of-the-art E2E correlation attack on collected datasets.

First we discuss the input format used for learning, which is the same as is used in prior work which evaluates the attack on the public dataset collected through a single proxy. We do not alter this representation. Our main evaluation methodology is applied as described in Section 4.5. We perform E2E attack replication on both datasets with single- and multi-proxy designs, which includes hyperparameter tuning and cross-validation with training and testing. Finally we report main performance results, scalability results on the bigger dataset, and the explainability analysis achieved through visualization of activation maps.

4.6.1 Experimental Setup

All deep learning models used in our experiments are implemented with Tensorflow version 2.2.0 and Keras version 2.3.0-tf. We modified the publicly available implementation of DeepCorr[6] to fit our experiments and implemented the replication procedure on top. The experiments are run on a server with 2 NVIDIA GeForce RTX 2080 Ti GPUs each with 12GB of RAM and an Intel Xeon CPU with 256GB of RAM.

Hyperparameter tuning is implemented with Tune [117] – a research platform for distributed model selection on top of Ray [137]. We used the HyperOpt [195] algorithm for model selection with Bayesian optimization [21]. We deployed a cluster on 6 Amazon EC2 P3 instances with 8 NVIDIA Tensor V100 GPUs each with 16GB RAM and an Intel Xeon CPU with 488GB RAM.

We make attack tuning, evaluation & analysis code available on our website provided in footnote 1.

Table 4.3: Architecture and tuned hyperparameters of the evaluated $\mathcal{TDC}$ models on single- and multi-proxy datasets. Fixed kernel sizes: (2, 30),(4, 10). Fixed strides: (2, 1), (4, 1).

Hyperparameter	**Search space**	$\mathcal{TDC}_{SD}$	$\mathcal{TDC}_{MD}$
kernels	[[64, 128],[64, 64] [128, 128]]	[64, 128]	[64, 128]
pool size	{3, 5}	5	5
dense layers	[[1000],[500, 100] [1000, 100]]	[500, 100]	[500, 100]
optimizer	—	Adam	Adam
lr	{0.001, 0.0001}	0.0001	0.0001
batch size	[32, 64, 128]	32	64
dropout	[0.0, 0.4, 0.6]	0.6	0.6
w_p	[1, 2, 5, 25]	5	5
N_{neg}	[9, 49, 199]	49	49

4.6.2 Data Input Format

Deep learning-based traffic correlation relies on automated feature extraction from a chosen representation of input traffic. In our experiments, we follow the prior work [145] in representing a bidirectional network flow as a time-series that

[6] `https://github.com/woodywff/deepcorr`

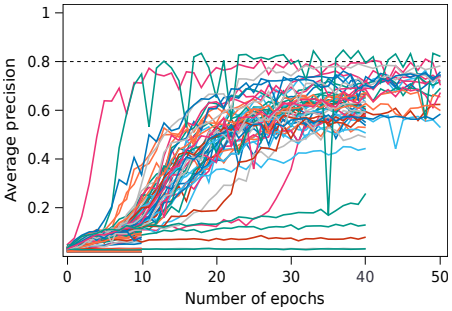
combines packet sizes, timing and direction features. Therefore, a traffic flow F is formatted as a time-series with 4 features: $F = [T^u, T^d, S^u, S^d]$, where T is inter-packet delays, S is packet sizes, and the u and d refer to the uplink (client to server) and downlink (server to client) directions. For all experiments reported in this study, we use maximum 300 packets in each of the 4 dimensions. Note that attack performance may increase with more packets included, however, to deploy our extensive evaluation methodology for comparing two E2E correlation datasets, one fixed length is considered representative. We also filter out traces with less than 100 packets in the uplink or downlink direction, as the prior work demonstrated that short traces impede the learning capacity of the attack.

Therefore, a deep learning attack attempts to correlate a pair of traffic segments F_i and F_j, where segment i is captured at a controlled Tor entry guard and segment j is captured at a controlled exit relay (or a proxy server in a research setting). And thus one input instance becomes of shape (8, 300): $F_{i,j} = [T_i^u, T_j^u, T_i^d, T_j^d, S_i^u, S_j^u, S_i^d, S_j^d]$.

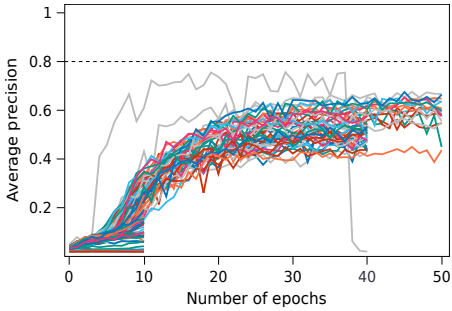
4.6.3 Attack Replication: Tuning

In this part of evaluation we zoom in to the Hyperparameter Tuning (HT) part of the Algorithm 4.1 (steps 10-18). We tune a given deep neural network individually on each dataset *SD* and *MD*. We explore two different kinds of search spaces: those informed by the original DeepCorr model $\mathcal{DC}$ (which we found to be visibly prone to overfitting), and those that drastically reduce capacity of the network to avoid this extra complexity. We call the reduced architecture $\mathcal{TDC}$ - Tiny DeepCorr. The main difference between $\mathcal{TDC}$ and $\mathcal{DC}$ is in the number of kernels used by each convolutional layer. We observe that decreasing the first layer from 2000 to 64 or 128 kernels, and the second layer from 800 to 128 kernels yields a significant multi-fold improvement in speed (9 hours training time down to 20 minutes, 11 hours testing time down to 40 minutes) and easier interpretable activation maps, while the average *AP* decreases by only $\approx 3\%$. Since for our evaluation we are mostly interested in reliable assessment of the relative performance of the attack on two datasets, we consider this performance drop justified. For transparency, we tune both models and report $\mathcal{DC}$ results in the Appendix B.1, but we focus the rest of the evaluation on the more optimal design – $\mathcal{TDC}$.

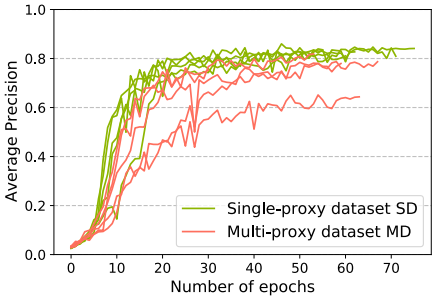
When tuning $\mathcal{TDC}$, apart from reducing the complexity of the network, we also vary the main learning and regularization parameters and the amounts of kernels and nodes for each layer. In order for the neural networks to process this input format in the way intended by DeepCorr, we preserve the original



(a) HT validation AP on the single-proxy dataset SD.



(b) HT validation AP of on the multi-proxy dataset MD.



(c) Validation AP per training epoch on each dataset.

Figure 4.6: Hyperparameter tuning $\mathcal{TDC}$: average precision (AP) of top-120 configurations over 50 epochs on (a) SD and (b) MD. Note that runs are terminated early when they do not progress well enough. (c) Cross-validation of $\mathcal{TDC}$ on both datasets with seed 10.

kernel width and the stride values in both convolutional layers. This way the timing and size features are analyzed independently.

Next, we vary the tunable N_{neg} parameter introduced by DeepCorr: it defines the number of non-associated pairs per each truly associated pair provided as negative examples to the network during training. A smaller N_{neg} results in less training data available to the network, but a bigger N_{neg} contributes to a more pronounced imbalance in the data between positive and negative samples, which also has an adverse impact on learning. In order to compensate for this imbalance, we propose to account for the imbalance in training data by using a *weighted* loss function. This is a known mechanism which modifies a learning process by augmenting the loss function with a predefined class weight value to penalize the error on the underrepresented class (truly-associated pairs). So

we tune the weight of the positive class w_p within the range from 1 (standard non-weighted loss) to N_{neg} (achieves a fully balanced loss).

The resulting search spaces for $\mathcal{TDC}$ are listed in Table 4.4 columns 1 and 2. We limit the number of experiments to $C = 200$ various configurations (combinations of an architecture and hyperparameters) for each dataset (and to 24 for the much more computationally intensive $\mathcal{DC}$). While exhaustive HT of deep neural networks is out of reach, the automated framework we use adopts Bayesian optimization to guide the search of best parameters. To decrease computational intensity of the algorithm, we run HT once on a subset of approximately $5,000$ traces (step 10).

Columns 3 and 4 in Table 4.4 report the final best configurations for single- and multi-proxy, while the top-5 configurations are listed in Table B.2 in Appendix B.2. As explained in Section 4.5, we select the optimal configuration based on the top achieved AP.

Figure 4.6 illustrates the training process for the top-120 runs on *SD* (a) and *MD* (b) in terms of *AP* on the validation data (convergence in terms of loss can be found in Appendix B.2). There are several conclusions to derive from the images. First, as expected, the loss and *AP* are not linearly related: at first *AP* grows with the minimized validation loss, but then often stabilizies even when the loss starts to increase again (indicating overfitting). The dual-optimization problem is not trivial, and finding the most optimal point in training the attack deserves further exploration. For our experiments, it suffices to select the optimal point in a consistent way for comparing single- to multi-proxy data.

The second and main observation is that in general, tuning on *MD* presents a harder challenge than tuning on *SD* with the same search space. We see that $\mathcal{TDC}_{SD}$ performance is predominantly higher, and the model reaches convergence on earlier epochs. Overfitting is also more present with *SD*. We also notice that some of the best runs for *MD* require more kernels in the first convolutional layer, showing a demand for higher learning capacity. All this is the first indication that the noise produced in the single-proxy setup may indeed increase feasibility of traffic correlation.

We denote the best configuration on the single-proxy data $\mathcal{TDC}_{SD}$ and on the multi-proxy data – $\mathcal{TDC}_{MD}$. While learning from the two datasets presents tasks of varying difficulty, the nature of the problem remains the same such that $\mathcal{TDC}_{SD}$ and $\mathcal{TDC}_{MD}$ are very similar: the only difference is in batch size. It is theoretically possible that further advanced tuning of $\mathcal{TDC}_{MD}$ could potentially improve the attack, e.g., by increasing complexity and employing additional regularization techniques to counteract overfitting. However, our HT experiment succeeds in showing that *the dataset generated in a more realistic*

multi-proxy setup presents a harder problem for learning.

Table 4.4: E2E attack performance of the tuned $\mathcal{TDC}$ models on the single- and multi-proxy datasets *SD* and *MD*. AP_{val} is computed on validation data of size $N \times N_{neg}$ ($15,895$ for *SD* and *MD*, and $50,000$ for MD_{1mln}). Other metrics are computed on the test set of $N \times N$ traces (2.5 million for *SD* and *MD*, and 25 million for MD_{1mln}), imitating actual attack. AP indicates the area under PR-curve and is thus not directly comparable between data with different skews [28], in this case the validation and test data. TPR, FPR and Pr are first optimized for a low $FPR = 0.001$ then for a high $TPR = 80\,\%$.

Model	$AP_{val}(\%)$	$AP_{test}(\%)$	$TPR_{FPR=0.001}(\%)$	$Pr_{FPR=0.001}(\%)$	$FPR_{TPR=80\,\%}$	$Pr_{TPR=80\,\%}$
$\mathcal{TDC}_{SD}$	79.22 ± 6.80	24.55 ± 6.98	47.76 ± 9.01	21.40 ± 3.27	0.0071 ± 0.003	7.68 ± 2.82
$\mathcal{TDC}_{MD}$	71.27 ± 9.84	13.83 ± 6.60	32.43 ± 10.59	15.02 ± 5.19	0.0154 ± 0.008	4.08 ± 1.98
$p-value$	**0.007568**	**0.000028**	–	–	–	–
$\mathcal{TDC}_{MD-1mln}$	95.3 ± 1.2	5.099 ± 1.22	35.93 ± 2.77	39.37 ± 1.75	0.775 ± 1.12	7.09 ± 2.03

4.6.4 Attack Replication: Training and Test

In the main stage of attack replication (steps 19-23 and 24-25 of Algorithm 4.1), we estimate and compare the attack performance on both datasets by using the whole amount of traffic traces and the best model configurations. Both *SD* and *MD* consist of $N = 7,948$ truly-associated pairs of traces. We perform 5-fold cross-validation ($k = 5$) where for each fold we train and validate on $80\,\%$ of correlated traces with $N_{neg} = 49$ negative examples each ($0.8N \times (1 + N_{neg}) = 317,900$) and test on $20\,\%$ of positive traces paired with each other ($0.2N \times (0.2N - 1) = 2,526,510$ traces). For each fold, we also repeat the evaluation $n = 3$ times with random initialization. The final result for each dataset is therefore averaged over $k \times n = 15$ runs, aiming to account for potential selection bias and non-deterministic behavior of deep learning models. Having more samples should allow to analyze significance of the results.

Same as with HT, we use AP for early stopping, i.e., we obtain the weights of a model that yield the highest AP (area under the PR-curve). The question is then whether these weights generalize well to the unseen data amounting $2,5$ million pairs of traces. Table 4.4 shows the final performance metrics (extended version can be found in Appendix B.2). Figures 4.6a and 4.6b depict the training process of every run for *SD* and *MD* in terms of AP and validation loss. Overall, E2E attack correlation is more successful on the single-proxy dataset: we observe the difference in AP means of $7.95\,\%$. For the same optimized FPR of 0.001, TPR on *MD* drops by $15.33\,\%$, while for the optimized TPR of $80\,\%$, FPR increases from $0.007\,\%$ to $0.015\,\%$, which translates to $100\,\%$ more

false positives. $\mathcal{TDC}_{\mathcal{SD}}$ also converges faster and appears to be more prone to overfitting.

The difference between means obtained through experiments is not informative on its own, as it could be due to random variations or chance. To show statistical significance of this comparison, we apply the two-sample Welch t-test (for unequal standard deviations) to compute the expected difference between two populations' means.[7] If according to this test the difference is significant (e.g., p-value lower than 0.05), we can assume even on basis of these limited observations that the observed trend would generalize well to other experiments and data splits. Namely, that the attack generally performs worse on *MD*. We apply the test to 40 test set AP values obtained by $\mathcal{TDC}_{\mathcal{SD}}$ and $\mathcal{TDC}_{\mathcal{MD}}$, achieving t-value= 4.866 and p-value= 0.000028, which indicate high statistical significance. In Appendix B.2, we shed more light on variability of performance on different dataset folds and random seeds.

Scalability. To assess how well the attack scales to a larger dataset, we evaluate $\mathcal{TDC}_{\mathcal{MD}}$ on 25,000 associated pairs from the MD_{1mln} dataset with $N_{neg} = 5$. The results in Table 4.4 indicate worsened performance on 25 mln test traces. The experiment illustrates the intrinsic challenge of E2E correlation attacks (cf. Section 4.5): the growing amount of intercepted traffic traces leads to a quadratic growth of the test set, which in turn causes a sharp decrease in the overall attack performance.

Results. To sum up, based on the extensive tuning and systematic evaluation, we can conclude that the E2E correlation attack performs *significantly better* on *SD* than on *MD*, which corresponds to our initial hypothesis. Namely, the realism issue with the single-proxy data collection setups that was exposed in Section 4.3 appears to favorably impact estimated performance of data-driven E2E attacks. As this may be due to the noisy timings present in the input, we further explore this connection with explainability analysis.

Explainability. To get the insight behind the observed difference in performance, we compute mean activations maps in the first convolutional layer, separately for correlated and non-correlated inputs, for both datasets. As explained in Section 4.5, the attacks may assign different levels of importance to various input features or certain areas of input traffic traces, indicating where the signal for traffic correlation is stronger in each dataset.

[7] Note that the Welch test assumes normality in measurements. We do not highlight what is the most suitable choice of a significance test for the given problem, as it largely depends on the metric and would require a focused investigation.

In Figure 4.7 we show one example of *the differences between mean activations* obtained from correlated and non-correlated pairs by single and multi-proxy models. Activations produced by $\mathcal{TDC}$'s first layer have shape $(4, 64, 271)$ (features, kernels, activation positions), and we plot the two computed timing features $[T^u_{ij}, T^d_{ij}]$ for positive and negative pairs. As the visual interpretation is limited, we also calculate a sum of the differences for each pair of models - the *total difference in mean activations* (see Table B.5 in Appendix B.4). The first observation to be made here is that in general, uplink timings appear to be more important to correlation than downlink timings. This is surprising, as downlink information contains server responses, i. e. is richer and of greater volume. As there are no studies that look into explainability of DL-based traffic correlation, more in-depth analysis is required to explain the mechanism through understand the exact involvement of kernels. Requests could facilitate E2E correlation if they were indeed more unique than responses. However, that does not have to be the case: DL-based nature of the attack may be causing the attention skew from downlink to uplink data simply because it is easier to analyze. Neural networks are known for simplicity bias: they can exclusively rely on the simplest features and remain invariant to other more complex predictive patterns. To verify this idea, we suggest that future work needs to investigate the inner workings of DL-based traffic correlation attacks, similar to the recent work for WF [51].

Secondly, we compare the activation differences of SD and MD and notice that on the single-proxy data, uplink timings are in general more impactful on the attack. This preliminary analysis indicates stronger timing signal in single-proxy traces; however, more research is needed to study the exact mechanism. We refer the reader to Appendix B.4 for more details.

Overall, the problem of reliable and robust AI evaluation remains an open research question. The guidelines proposed in this chapter address common sources of sampling and evaluation bias in experimental setups and can be further improved in future research.

4.7 Discussion

Here we discuss limitations and future work directions.

Representativeness of Traffic. Our dataset consists of website requests that only consider visits to the main homepage while omitting the pages within. This simplification allows us to create a dataset comparable with prior work on E2E correlation, so the main quality difference is manifested through much more

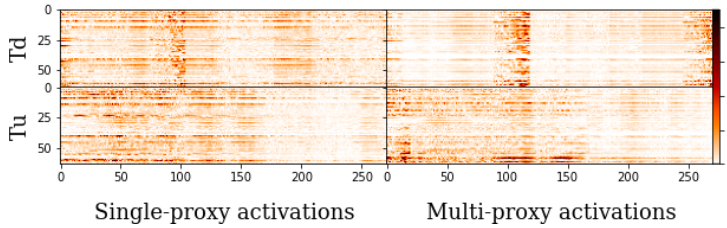


Figure 4.7: Differences between mean activation maps computed over 1590 correlated (pos) and non-correlated (neg) traces. Left: single-proxy, right: multi-proxy. Datasets fold: 4, seed: 10. Uplink timings: T^u, downlink: T^d. Dark color indicates higher intensity.

accurate temporal characteristics. However, we did not aim to move the dataset beyond the state-of-the-art in terms of representativeness of simulated human behavior. While our specific analysis of E2E attacks remains largely undisturbed by this limitation and yields novel results, the browsing activity stays within the abstraction of homepage visits, as is common for many other known WF and E2E datasets [145, 170, 193, 194, 151]. There have been notable advances in the area of approximating real Tor user profiles and addressing complex website infrastructures during traffic generation for WF [154]. Using our novel setup to collect an E2E (and WF) dataset that is also more representative of human behavior and assess the impact of traffic analysis attacks would be a challenging but valuable extension.

Proxy-Related Overhead. A proxy-based data collection setup for end-to-end correlation allows to estimate attack success rates in a research setting without compromising safety of Tor users. In this study we show that the novel multi-proxy data collection setup reduces the timing overhead introduced by the state-of-the-art single-proxy design. However, the impact of *any* involvement of proxies on traffic metadata, in comparison to the real-world Tor, remains an open research question. The complex proxy-related overhead and its indirect impact on traffic correlation feasibility can technically only be measured by collecting real exit traffic. As this is associated not only with the loss of node location diversity but also with strong ethical concerns, it would be pertinent to minimize privacy risks by consulting the community for coordination of effort (e. g., through the Tor Research Safety Board.[8])

[8] `https://research.torproject.org/safetyboard/`

Ethics Considerations. Work on traffic analysis attacks requires realistic datasets that resemble Tor's daily transmission characteristics. Measurements in the real network must not harm or deanonymize Tor users by any means or affect the overall performance of the system. For all decisions we made when designing our real-world experimental setup, we closely followed Tor's guidelines for ethical research [160]. Although frameworks like the Shadow simulator [101] are able to mimic transmissions through the Tor network, we decided to analyze real-world traffic to allow for a fair comparison with previous research.

Defenses Against Traffic Analysis. The analysis in this chapter is focused on unprotected traffic. While this is sufficient to thoroughly validate our proposed data collection setup, evaluation of known protection schemes is a worthwhile future research direction. To this end, we make our dataset on closed-world and open-world WF and E2E attacks publicly available to foster research on a joint evaluation of defensive schemes against both attacks. To the best of our knowledge, our dataset is the first effort in combining both threat models in one data collection setup, enabling this type of large-scale exploration.

Onion Services. When browsing a typical website, it could be indicating that it can be accessed over an onion service, using the `Alt-Svc` response header. In our setup, the targeted websites observed incoming connections from our proxies, hosted on the network of a cloud provider. Consequently, the websites would not send the `Alt-Svc` header, and thus our Tor Browser client did not attempt to access the them over the onion service. Moreover, because the Tor Browser used a proxy chain, it would not be able to resolve or connect to `.onion` addresses. Consequently, our data collection setup deviates from a real-world data capture for websites that would redirect users to an onion service.

In practice, we find that Cloudflare, a large CDN provider, provides an optional feature for all the websites it hosts that allows them to be accessible via an onion service [178]. However, because of our proxy-based setup, no traffic was sent to onion services, possibly affecting the representativeness of our dataset. More specifically, we captured data at our proxy that might have otherwise been sent to an onion service, and thus would not be captured by an adversary who only controlled the client's guard node and the exit node. In a real-world attack, the adversary needs to take the asymmetry of requests sent to exit nodes and onion services into account. In preliminary tests, we found that onion services are rarely contacted when loading a new website. Nevertheless, an extensive evaluation of the impact of loading resource from onion services on attacks is an important aspect to be addressed in future work.

4.8 Related Work

Our work builds on DeepCorr, the first traffic correlation attack that leverages deep learning [145], as opposed to the previously attempted statistical approaches (such as its closest competitor RAPTOR [203], which utilizes the Spearman rank correlation metric and only achieves good performance in a limited setting of 50 targeted connections). While DeepCorr undoubtedly outperforms previously proposed approaches to flow correlation, we critically replicated its evaluation focusing on a more realistic research setup for modeling the attack. Our study provides additional novel insights on performance evaluation, tuning, scalability and explainability of the DL-based attack. Concurrently to our work, Oh et al. [67] have substantially improved on DeepCorr by utilizing metric learning techniques. As their learning approach is fundamentally different from CNN in DeepCorr, an interesting research direction would be to investigate performance of their attack in our proposed multi-proxy setup.

Previously, critical evaluations of attack techniques in the light of their real-world applicability have been conducted in the context of Website Fingerprinting [104], with focus on access to traffic as operational requirements for end-to-end confirmation [103, 180], and for de-anonymization attacks targeted at specific users [100].

As a widely adopted technique for end-to-end processing of time-series, DL is favored for unprecedented performance and efficiency. With the analysis of Tor traffic being one of the most prominent applications in privacy [152, 170], DL was also employed, e.g., for protocol detection [229], encrypted video stream analysis [182], and for inferring (mobile) applications from encrypted traffic [3]. Limitations and challenges of DL evaluations in security have been recently explored by Arp et al. [13]. However, most progress on sound practices for DL evaluations are conducted in the AI domain [140, 39, 50].

4.9 Conclusion

In this chapter, we introduced new methodologies for data-driven end-to-end correlation attacks on Tor. We proposed a new experimental setup that allows to collect Tor traffic with more realistic timing characteristics by minimizing the additional overhead in proxy-based end-to-end measurements. Furthermore, we introduced a systematic replication strategy along with appropriate evaluation metrics to allow for a fair comparison of data-driven attacks on novel data. Our empirical results demonstrate the relevance of the suggested multi-proxy design: we find that the novel end-to-end correlation dataset that contains more realistic

timing measurements also presents a significantly harder learning problem. This indicates that utilizing multiple proxies approximates attacker capabilities more accurately than prior work – a positive step towards realistic attack evaluation. Finally, we open source the research code and the multi-proxy dataset to enable further research on traffic correlation and website fingerprinting attacks.

Part II

Network Intrusion Detection

Chapter 5

Data-Driven Network Intrusion Detection

Preamble

This chapter continues the discussion of enhancing cybersecurity by applying AI techniques, here in the scope of network intrusion detection. First, we revisit the foundations of machine learning-based solutions for network security, which generally aim to make network defense tools more autonomous, adaptive, proactive, and responsive. Then, along with outlining recent advances in this broad field, we reflect on some fundamental limitations and the associated research challenges.

Parts of this chapter have been previously published in "Security and Artificial Intelligence" [169] as a book chapter focusing on open-world network intrusion detection. In this thesis, the text from the book chapter appears either verbatim or with modifications and serves as the background for the subsequent Chapter 6 on deep learning for network intrusion detection.

5.1 Introduction

Intrusion detection is an integral part of securing information systems. Detection tools stem from the early realization of the computer security community that full and provable protection of an ICT infrastructure is practically infeasible, if

not impossible. Attempts to compromise the system can emerge from within the infrastructure as well as from its adversarial environment. An Intrusion Detection System (IDS) therefore aims at detecting exploitation attempts and active misuse both within the perimeter and from outside of it.

Continuous monitoring of the system and accurate detection of malicious behavior constitute the first step of the incident response process. For this fundamental step, an IDS performs acquisition of relevant data streams that represent operation of the system and its internal and external communication. Through an in-depth real-time analysis of these monitored data, an IDS searches for any signs of a potential misuse of the system. If such evidence is detected with high enough confidence, an alert is raised which is then propagated to security analysts, or to a Security Information and Event Management (SIEM) system. SIEM systems are used mostly in larger networks to aggregate and correlate alerts and other security events. This way the incident response team can analyze the data from IDS and other sources (network logs, firewalls, servers, antivirus software, etc.) in one place to construct a full picture of threats in the network before taking measures.

In essence, an IDS is responsible for intelligent automated decision making that can, depending on the correctness of those decisions, either safeguard or disrupt normal operation of an infrastructure. Reliable intrusion detection is indispensable, but despite the world-wide efforts of the last 40 years, IDSs regularly generate false alarms, at times fail to prevent intrusions, and thus end up jeopardizing system and data security with high recovery costs. A vast amount of research and development has gone into creating dedicated tools and algorithms in order to bring intelligent and reliable intrusion detection into reality. Among the possible solutions, Artificial Intelligence (AI) has always been a compelling component for automated knowledge retrieval that aids in efficient detection of ever-changing attacks, with a varying level of complexity and involvement. Especially in light of recent advances in the Machine Learning (ML) and Deep Learning (DL) domains, the solution space is evolving so rapidly that it has become challenging to keep track of major changes.

While AI algorithms are ubiquitously used across all possible data sources and points of IDS deployment, in this chapter we focus on network intrusion detection – one particularly prominent and highly representative application of machine learning in network security. Network IDSs use network traffic as the primary data source for detection. They can detect attacks either through *misuse detection*, i.e., matching observed traffic to a known malicious traffic signature, or through *anomaly detection*, i.e., detecting suspicious patterns as indicators of malicious activity by comparing traffic to a previously established benign baseline. These two strategies target different objectives and present

unique research challenges An anomaly detection system (or a behaviour-based IDS) does not rely on the knowledge of exact attack signatures, but instead detects earlier unseen patterns. By respecting the *open-world* nature of intrusion detection, where attacks are continuously evolving, anomaly detection thus carries an intrinsic potential to discover novel attack types. Misuse detection is targeted against known, widespread attacks and their (close) variants. As opposed to anomaly detection, it typically assumes a closed set of possible attacks that it can recognize. Both strategies can be employed in the same infrastructure as components of a comprehensive defensive solution. Furthermore, both can benefit from advances in machine learning to gain stronger capabilities. For instance, machine learning can transform a misuse detection system (or a signature-based IDS) to an open-set recognition, to be able to explicitly identify a pattern it has not seen before. Together, machine learning-enabled misuse detection and anomaly detection have a potential to augment the capabilities of existing security tools and are widely utilized in industry.

The chapter is organized as follows. In Section 5.2, we introduce the IDS domain and its core concepts. After reviewing the problem statement and the threat model of a network-based IDS, we move on to outlining the principal machine learning techniques and their underlying assumptions in Section 5.3. Finally, in Section 5.4, we analyze general open challenges for real-world deployment and for research on network intrusion detection. We conclude our reflections in Section 5.5.

5.2 Network Intrusion Detection

Malicious attempts to invade an ICT infrastructure must be detected and localized while in progress. This objective implies performing real-time analysis of continuous streams of data from various sources and locations in search for indicators of compromise. Traditionally, the process of intrusion detection has been split between Network Intrusion Detection Systems (NIDSs) and Host Intrusion Detection Systems (HIDSs). Both serve complementary purposes by monitoring malicious activities at different levels. HIDS runs on internal nodes, carrying the ability to closely monitor their individual behavior. It primarily relies on host-specific data sources, such as system and kernel-level activity traces, program analysis, audit logs, files and documents. Maintaining a fine-grained access to activities on individual hosts enables a HIDS to precisely localize misuse, e.g., an active malware. For further information on HIDSs and

the diverse data types they utilize to detect attacks, we refer the reader to the recent survey [29].

The holy grail of intrusion detection is to recognize threats as early as possible before the system and its data get compromised. Since the most common way for intruders to enter an infrastructure is through the network, a NIDS – being placed at the edge of the network – analyses inbound and outbound network traffic and thus acts as the first line of defense. As opposed to the host-based level of monitoring, a network-based approach gives a more expansive view over the network, allowing for an early detection of attacks, including those that target multiple hosts at once. NIDSs can be also freely deployed within the perimeter, enabling them to monitor traffic traversing inside the network and to detect internal attacks. Today, security is moving beyond traditional perimeter-based solutions: the growth in cloud, mobile and edge computing and connected devices makes network borders and access points much less defined. The *Zero Trust* paradigm has been introduced [110] to address the gaps in traditional network design by (i) not trusting any entities inside or outside the organization network, and by (ii) segmenting the network and thus providing only limited access required to perform specific tasks, even after verification. From the perspective of network intrusion detection, this entails that all network traffic is treated as untrusted, increasing the workload for NIDSs deployed at choke points in the zero-trust network.

AI-powered traffic analysis and network defense tools are becoming ubiquitous across ICT infrastructures, being adopted to diverse network architectures, points of NIDS deployment and data sources. Before diving into the data-driven approach used to enhance network intrusion detection, we establish a general threat model under which a NIDS operates.

5.2.1 Network Threats

Cyberthreats targeted by a NIDS are either executed over a network, or communicate with external parties over a network. A NIDS mirrors the entire network traffic that is permitted by a firewall, being able to distinguish attacks across the OSI stack, from Layer 2 (Data Link) potentially all the way up to Layer 7 (Application). Very broadly, these threats can be categorized into passive and active attacks.

Passive attacks do not involve any meaningful interaction with nodes in a target network and do not alter any data. Instead, the purpose is to *probe* the system – obtain useful information which can be efficiently collected through, for instance, *network scanning* or *port scanning*. Network scanning allows to detect accessible nodes, while probing open ports allows to identify services running on these

nodes, exposing their vulnerabilities. As such, a passive attack does not leave any traces in the system and is by itself a preparatory step before a more aggressive intervention. Detecting a passive attack in real-time gives a defender an opportunity to proactively identify an adversary and prevent their intrusion.

The types of *active attacks*, however, vary greatly. These are the network attacks that aim at compromising integrity, confidentiality and availability of target systems. Without being exhaustive, we list some of the most prominent attack scenarios.

The most basic way to *penetrate* the network would be by brute forcing credentials of a legitimate user, which is characterized by an overwhelming series of unsuccessful logins rather noticeable in the inbound network traffic. More advanced attackers penetrate a protected network by carefully exploiting vulnerabilities found in its perimeter and thus gaining unauthorized, potentially privileged access to the system in a more stealthy manner. These can be misconfigurations or vulnerabilities in firmware of entry-level network devices, or web and software vulnerabilities of publicly accessible hosts, such as buffer overflows, Cross-Site Scripting (XSS) and SQL injections. Similar to brute force attacks, these exploits can also be launched over the network, carried in incoming network packets' payloads. However, due to traffic encryption and a number of other considerations to be discussed further, malicious payloads even of well-known exploits are not guaranteed to be detected.

Apart from penetration attacks, another large family of network threats are Denial of Service (DoS) attacks, which aim at disrupting normal functioning of target hosts and deny their availability to legitimate users. This goal can be achieved through flooding the victim node or resource with superfluous requests in an attempt to overload both the network bandwidth and the system, possibly also targeting its IDS. There are many variations of a DoS attack, including SYN flooding, ICMP flooding, smurf and others, which differ in mechanics and final effects. A Distributed Denial of Service (DDoS) is launched from numerous sources at once, often automated by a whole network of compromised computers – bots. Today, *botnets* are seen as the largest network security threat and remain one of the key research topics in intrusion detection. Bots become disguised as legitimate actors through infecting privately owned systems. Their automated illegitimate activities can have various malicious impact, ranging from spam and click fraud campaigns to identity theft, DoS attacks and malware infections. Remote Command and control (C&C) servers send instructions to the compromised cluster of computers and receive back reports and leaked information. Even though botnets generate a lot of communication, detecting them through traffic analysis or other means is a serious challenge for any type of IDS, as bots continue to evolve and find better disguises.

While botnets are rather stealthy and can cause extremely damaging consequences, there is an attack class that surpasses others in evasiveness, sophistication and severity – Advanced Persistent Threats (APTs). These threats are human-driven attacks targeted against a specific infrastructure and aimed at gaining an ongoing access for a long period of time in order to exfiltrate sensitive valuable data – this could be, for instance, intellectual property of organizations, trade secrets, or customer information. It is especially difficult to detect an APT at the moment of perimeter penetration or privilege escalation, since advanced attackers may use unknown exploits or social engineering tricks to infiltrate an infrastructure. Afterwards, an APT only infects a few chosen hosts to get closer to valuable resources of the network, effectively staying under the radar. They rarely contact remote C&C servers, and when they do, they use encryption or obfuscation techniques, complicating traffic analysis by a NIDS. However, the *data exfiltration* process – a common goal of an APT – may be well observable in outbound network traffic, which grants a crucial defensive role to a NIDS.

5.2.2 Network Traffic Monitoring

Network traffic became a universal data source for intrusion detection thanks to standardization and ubiquity of network protocols, which makes NIDSs adaptable to a wide range of platforms and applications. A NIDS collects and inspects network traffic in different modes, mostly either on a packet level or on a flow level. These approaches to traffic monitoring differ in informational content and practicality, and selecting one, or a combination, depends on the environment and the threat model.

Packet-Level Inspection. Capturing traffic from the network by a NIDS for monitoring and analysis purposes is achieved with promiscuous access to *copies* of network packets, and therefore without interference in communication. Full packet captures are usually made in the *pcap* file format, a widely used and portable format for packet inspection. Pcap files can be processed with Deep Packet Inspection (DPI), which performs both packet header and payload analysis. DPI can provide extensive information about communication, exposing malicious payloads to a NIDS. One of the earliest IDS tools that performs application-level DPI is Snort [173], an open-source signature-based detector. Snort matches observed packets with known malicious patterns using regular expressions (e.g., for a Linux web server, a pattern could be an HTTP request containing "`/etc/passwd`"). A more recent open-source NIDS called Suricata [205] improves the scalability of Snort [6]. Such signature-based NIDS strongly rely on a rich database of malicious payloads. For anomaly-based NIDS,

one prominent example is the Zeek [237] tool (formerly known as Bro [158]). Zeek constructs benign baseline profiles for an application's usage based on predefined policy scripts and flags deviations from these profiles. For instance, it was shown to effectively detect web attacks, such as reflected XSS injection and SQL injection, by inspecting strings in the HTTP-request parameters [212]. With access to payloads, these attacks are straightforwardly detected due to presence of unusual characters in the request body.

Despite all these advantages, processing full packet captures comes with some considerable practical issues. The sheer volume of packets in modern high speed communication networks is overwhelming, making DPI inefficient or even infeasible in real-time. Moreover, storing full captures for further network forensics is a highly limited resource. Captures of very large packets are often incomplete or even limited to header information, largely omitting the most informative parts. Finally, two fundamental limitations to performing DPI are (i) invasion of privacy through accessing and storing benign packets' content, and (ii) traffic encryption. Packet-level inspection on TLS-encrypted traffic can be realized through man-in-the-middle solutions that decrypt and re-encrypt payloads, thereby violating end-to-end security guarantees, which can be both unsafe and computationally intensive for a particular environment. An ongoing line of research explores DPI over encrypted traffic through matching encrypted tokens with encrypted rules [187, 150]. Currently this approach requires computationally intensive setup phases for every network connection, and, without the aid of decryption, supports only a limited number of IDS rules.

In view of the above circumstances – high data rates, computationally demanding processing, privacy and encryption concerns – DPI for intrusion detection is arguably becoming increasingly obsolete in modern environments. Nevertheless, there exist numerous network intrusion detection datasets with full packet captures, as packet-level analysis has proven to be highly beneficial for research purposes.

Flow-Level Inspection. Rather than inspecting and storing all individual incoming and outgoing packets, a NIDS may group relevant packets together in a *flow* and collect their aggregated information on a flow level. A traffic flow is commonly defined as a series of bidirectional packets exchanged between two hosts that share a five-tuple – source and destination IP addresses, source and destination ports, and a protocol – collected over a certain period of time. This may correspond to a duration of one complete network connection, or to a predefined time window (until a timeout is reached). Flow-level information is aggregated across all the packets belonging to one flow – typically this includes timestamps of start and finish, number of packets and bytes, arrival times,

certain packet header attributes, etc. These traffic flow metadata provide a high-level description of communication between source and destination hosts which can be very telling about its benign or malicious nature [198]. As a result, flow-based inspection does not take into account traffic content, but instead reveals informative high-level communication patterns, while greatly reducing the size of data to be analyzed.

Packet counts are recorded when packets cross network router interfaces. Most often, flow data aggregation is performed through Netflow [44] – a network monitoring protocol that is well integrated in modern network environments. Since Netflow counters are mostly generated directly on the network equipment, performance of the network may be affected. The overhead can be limited by performing traffic aggregation at the hardware level or even by decoupling traffic routing and flow computation by passively copying traffic data, similar to the case of packet inspection. As opposed to the packet-level inspection, which mostly provides signature-based analysis, an advantage of flow-based inspection is that it supports an anomaly detection approach, as we describe in Section 5.3. Furthermore, network aggregates are applicable in the context of end-to-end encryption or privacy constraints of a particular environment, because they omit packet payloads from analysis. This property facilitates public availability of real traffic data aggregated in the form of flows, which is extremely valuable for open network intrusion detection research. A NIDS can also implement a hybrid approach that combines DPI and traffic flow analysis in a number of variations, mostly relying on the flow-based analysis complemented with an occasional payload inspection of suspicious traces.

5.3 A Data Analysis Approach

Monitored traffic traces – in the form of full captures or aggregated flows – are analyzed to find indicators of potential attacks. In the early days, review of monitored activities for intrusion detection was performed manually by security analysts or system administrators. They used to devise and manually adjust rules and heuristics that would help to find harmful packets and identify suspicious behavior. The volume and increasing complexity of monitored data has long deemed any such manual efforts insufficient and prompted the community to introduce automation. Already in the 1990s, progress in AI research enabled investigation of ML techniques in application to intrusion detection. The power of data analysis is in interpreting large amounts of data and automatically discovering new relevant knowledge – a highly valuable capability in the ever-growing and ever-changing security landscape.

A ML-based IDS employs a data-driven approach to intrusion detection – it uses ML methods to autonomously learn characteristic rules and patterns from previously observed data. For the case of network intrusion detection, the abundance of network traffic data creates an opportunity to apply data-driven techniques. A ML-based NIDS configured for *misuse detection* can detect variants of known attacks by finding patterns sufficiently similar to previously seen malicious traffic. On the other hand, a ML-based NIDS that performs *anomaly detection* can model "normal" behavior of the system by learning from benign network traffic, and catches anomalous patterns that deviate significantly from the baseline. In the context of high non-stationarity and strong heterogeneity of network traffic, another strength of ML is its ability to dynamically adapt to changes in the network when exposed to new data.

The prolific use of AI cannot be merely attributed to impressive automation capabilities of ML algorithms, as it also relies heavily on expert involvement. In order to benefit from the advantages offered by ML, the designer of a ML-based NIDS applies their domain expertise to create an appropriate learning system. First and foremost, an in-depth understanding of the network environment and the threat model are required in the *data representation* phase that converts monitored traffic data into a suitable format for ML. Applying ML methods directly to raw monitoring data, such as full pcap files, is not only hardly computationally feasible, but also does not usually yield useful results. The reason is that the numerous values in their original form are not equally relevant to the learning problem, which is especially true for data of such complex structure and overwhelming volume as network traffic. Therefore, an IDS designer leverages expert knowledge to find a compact representation of raw traffic which conveys characteristics that are most relevant to the task of misuse or anomaly detection. This step is known as *feature extraction* – a transformation that converts an initial set of input data into derived values, or *features*, that capture its underlying structure and complexity. As a result, information-rich raw data is represented as a *feature vector* containing compact, non-redundant information that is appropriate for subsequent learning and inference. The goal of the feature extraction process is to compose such features that describe patterns in data which, during inference, can *generalize* to previously unseen samples drawn from the same distribution. Today there exist two major general approaches to extracting features: (i) *feature engineering & selection* – hand-crafting and selecting most salient features based on practical experience and intuition about the problem, and (ii) *feature learning* – automated feature extraction with the use of learning algorithms.

The quality of features constructed from input data is one of the most influential factors that define the effectiveness of a ML algorithm. For optimal performance, it is crucial to keep the feature set up-to-date with the ongoing changes in the

learning problem – a concern especially present in network security. Therefore, in NIDS domain, both approaches to feature extraction are continuously revised and improved: while the former has traditionally played a predominant role, the latter is receiving increasing attention lately in light of recent progress in DL research.

Feature Engineering and Selection. When engineering features for network intrusion detection, and other related traffic analysis tasks [27], it is important to consider the characteristics of the learning problem and define the level of granularity of features. In networking, an analyst can choose from the wide range of packet-level, flow-level, session- or connection-level, and multi-flow features, with increasing level of granularity. On each level, a statistical description of data, such as distributions of attributes or groups of attributes, can be formed:

- *Packet-level* features are extracted or derived from collected packets: payload information, packet-specific header attributes and packet-level statistics, e.g., mean and variance of packet sizes and inter-arrival times.
- *Flow-level* description includes simple aggregated statistical features such as mean flow duration, mean number of packets per flow, mean number of bytes per flow, bytes transmitted per second, and more. The latest versions of NetFlow compute very basic features of traffic flows in real time, including bytes per packet and packets per flow, which are further used to derive flow-based statistics. Despite the simplicity, statistical features turn out to be highly informative of the nature of a particular communication, as they are quite effective in revealing traffic anomalies and particular known malicious patterns. For instance, a basic DoS attack is a volumetric attack that is commonly characterized by sending many packets in one direction within a short time period, making flow statistics well-suited for detection.
- *Connection-level* features are extracted from the transport layer and include information about the particular network connection (which may be split into several flows). For TCP connections, these additional features may include advertised window sizes in TCP headers and the throughput distributions.
- *Multi-flow* features appear especially valuable for security applications such as network intrusion detection. For some types of intrusions, meaningful revealing information can be only derived through aggregation across multiple flows/connections. For instance, for a particular flow, the number of recent connections from the same source or to the same destination can be relevant. These aggregated measurements are highly instrumental, for

instance, in detecting attacks executed over multiple connections, be that from different sources against the same victim, or against different nodes in the network. Probe attacks, DDoS attacks and bot communication are obvious examples, where one flow in isolation might appear harmless, while the overall behavior is more indicative. *Temporal statistical features* is another example of feature extraction approach that aims to solve the problem of a narrow one-flow view by aggregating traffic information over time across multiple flows, thereby respecting temporal dependencies between them [133].

The set of engineered features is often further optimized through automatic *feature selection* in order to discard redundant or irrelevant features and reduce dimensionality of voluminous data. In this way, feature selection may decrease computational overhead while maintaining accuracy, and also help to avoid *overfitting* of a ML model to the training data of high complexity, i.e., memorizing the data instead of deriving the underlying patterns. For different approaches to composing and reducing the feature set, we refer the reader to the corresponding surveys [53, 148, 27].

The resulting new set of high-quality manually extracted features forms feature vectors that are compact, non-redundant, informative and generalized, and possibly *interpretable*, having clear semantics for each vector dimension. Arriving at high quality representations is a challenging iterative manual process, involving experimentation with many techniques in combination with domain expertise and intuition about the problem. *Feature learning* aims at automating the process of feature extraction, relaxing the need for close expert involvement.

Feature Learning. As ML approaches struggle with high-dimensional inputs and manual feature extraction, *deep neural networks* have recently been embraced for dimensionality reduction and feature/representation learning. DL approaches are able to automatically extract discriminative internal representations of the input through a series of non-linear transformations upon observing sufficient amounts of data. Several studies applied *deep belief networks* for misuse and anomaly detection in order to obviate manual engineering of traffic flow features [73, 66, 7, 189]. The literature has shown that automatically learned representations seem to be more robust to irrelevant deviations in data and thus should contribute to higher generalizability of ML models to earlier unseen patterns. The immediate drawback of automatically learned abstract features is that, as opposed to hand-crafted feature vectors, they do not provide clear semantics. Decisions of DL-based NIDS are, therefore, hardly directly interpretable. This calls for additional methods to verify that a trained DL-based feature extractor has learned appropriate meaningful traffic patterns and

to explain a model's decisions to security analysts [8, 149]. Therefore, more automation implies less interpretability – a trade-off that a NIDS designer has to balance.

There are additional concerns regarding the usage of deep neural networks for network intrusion detection, in terms of both real-world deployment and applied research. We elaborate on the problem of feature learning with deep neural networks for network intrusion detection in Chapter 6.

5.3.1 Machine Learning for NIDS

When selecting an appropriate ML model, it is crucial to understand how to leverage the properties of features and the relationships between them. For instance, streaming data represented as a time series consists of temporal features, which are best interpreted with a model capable of recognizing temporal dependencies. The choice of a learning algorithm, however, starts with defining a concrete ML problem statement that most accurately represents the task of intrusion detection, be it misuse- or anomaly-based. This encapsulates at the very least such influential factors as (i) the expected input and output of the system, (ii) the assumption about knowledge of all existing data categories, and (iii) availability of annotated training data.

Input and Output. The expected input implies the data representation, i.e., types and dimensionality of extracted and selected features. The expected output of the model is a design choice of how to present the inferred information about the event for further analysis and response. In general, for a given test instance, the output can be a *label*: malicious vs. benign or anomaly vs. benign. For anomaly detection in particular, it can also be an anomaly score that indicates significance of the detected anomaly for further investigation. Optionally, the model can also provide its confidence score for each decision.

Closed-World and Open-World Assumptions. The assumption about knowledge of all data categories is what largely drives the choice between misuse and anomaly detection in the first place. A misuse detector is typically deployed under a *closed-world* assumption, which implies that all possible data categories, i.e., types of intrusions, have been seen at the training stage of the model. A common closed-world NIDS employs a ML *classifier* that learns to recognize a traffic instance as benign or belonging to one or another attack class, thus performing *intrusion recognition*. Closed-world detection has been thoroughly researched in the network security domain, and has been traditionally favored by

industry due to predictability and high detection rates. However, in operation, such a model can only detect known malicious behaviors and assumes that no unexpected attack type may appear.

In reality, a network environment operates under a much more challenging threat model that includes known attacks, new variants of known attacks and completely novel, earlier unseen cyber-threats, comprising an *open world* of possibilities. In order to enable open-world detection, a ML problem statement has to change from standard classification to either *open-world recognition* [20] (also called open-set recognition/classification), or *anomaly detection* [37] (also called outlier detection). An open-world classifier performs its originally intended task, but also leverages additional mechanisms to be able to identify novel patterns as instances that cannot be confidently classified as one of the learned attack types or as benign. Anomaly detection algorithms, however, are inherently open-world: as was explained earlier, anomaly detection exclusively relies on knowledge of benign data (normal, background traffic), and flags any sufficiently deviant pattern as a potential intrusion attempt of unknown nature. Therefore, in operation, an anomaly-based NIDS is an open-world detector as it targets both known and unknown attacks, although it can similarly use patterns of earlier seen attacks for model evaluation at the design stage. A core drawback of employing anomaly detection for defensive purposes is its dependence on the notion of "normality" – as a consequence, a pattern that deviates from normal data for benign reasons is also flagged as potentially malicious, usually causing a high number of false alerts that need to be investigated.

There are many studies that join anomaly detection with misuse detection in an attempt to combine the strengths of both paradigms: improve the detection rate and minimize the rate of false alerts. In fact, commercial platforms very rarely use anomaly detection in isolation, but rather adopt the *hybrid* approach [109]. The mismatch between the promises of anomaly detection and its actual adoption in industry is what demands a more explicit academic focus on anomaly-based IDS research.

Supervised and Unsupervised Learning. Another fundamental distinction between ML approaches relates to availability of annotated/*labeled* data. *Supervised* ML, such as classification, is a learning mode that relies on labeled training data. Namely, a classifier requires a significant number of representative labeled training examples from all the considered classes. Therefore, a classifier-based misuse detector works with a continuously updated database of known malicious patterns that need to be well represented in training data for a NIDS. This also applies to open-set classifiers. Acquiring labeled malicious data is expensive, as it requires either manual investigation by network experts, or

development of automated labeling algorithms, which essentially create a chicken and egg problem. Moreover, supervised learning is in general highly sensitive to *class imbalance* in data, demanding equal representation of every class. Otherwise, a classifier trained with imbalanced data becomes biased towards the majority class and largely ignores instances belonging to the minority class of interest. Since in network environments benign traffic is predominant, the benign class outweighs malicious traffic classes in labeled training data, causing a much lower representation of attacks. This undermines the sole purpose of intrusion detection, since the most interesting and inherently rare intrusions become overlooked. The issue can be addressed by using specialized techniques to increase importance of the minority attack classes [209]. Another solution is attack simulation performed to generate more malicious traffic for training under an assumption of its representativeness of real intrusions. However, for simulated benchmark datasets, the class imbalance problem is not unheard of, either.

Unsupervised ML paradigms obviate the need for labeled data. A general example is *clustering*, which performs exploratory data analysis to draw inferences and find hidden patterns and correlations in unlabeled data. Clusters are automatically formed with the use of a similarity measure between instances. Unsupervised approaches generally do not assume any a priori knowledge on the data distribution and labels, which corresponds to a realistic NIDS scenario. In practice, though, *semi-supervised* anomaly detection is often applicable under the assumption of availability of labeled normal data. As the shortage of malicious labeled real traffic is the main issue, both unsupervised and semi-supervised approaches pose the biggest interest for NIDS research.

While supervised NIDS approaches are widely utilized and thoroughly studied in the literature, this is mostly because of the convenience of evaluation and comparison between different algorithms on labeled benchmark data. Unfortunately, however, supervised NIDSs either violate the open-world context of network security, or extensively rely on manual data labeling, or both. In recent years, unsupervised and semi-supervised techniques in application to NIDS are gaining more traction; however, the research on their applicability in real environments is largely ongoing.

The community has composed a number of excellent surveys on the topic that provide detailed taxonomies and analyses of existing ML-based approaches to NIDS. For the closed-world misuse detection research, we refer the reader to the corresponding expansive literature that surveys supervised classification methods [31, 22]. Unsupervised or semi-supervised ML paradigms, which can enable open-world detection, are described in existing surveys [74, 207, 4, 69], which provide in-depth analyses of individual techniques, with a broad overview of the current solution space.

Further in this chapter, we reflect on some of the key challenges of NIDS research, including general challenges or specifically for the more relevant open-world detection.

5.4 Challenges and Advances in NIDS Research

Machine learning algorithms, and anomaly detection in particular, have gained a lot of attention in network intrusion detection research because of their compelling potential in detecting novel attacks. A decade ago, the community brought into the spotlight the intrinsic challenges of open-world network intrusion detection [76, 196, 207, 74]. It turned out that most of the conducted research explored ML-based IDS solutions under numerous unrealistic assumptions. In reality, with these wishful assumptions dropped, the effectiveness of ML-based solutions in detecting novel and known attacks falls way down below the estimated performance. In the context of a NIDS, ML algorithms are tasked with searching for the unknown, while costs for mistakes in a security-critical environment are high. A fundamental question was raised as to how appropriate ML algorithms are to such defensive applications, and which guarantees they can give for operation in sensitive environments.

Since then, the security domain grew significantly, with attacks becoming more sophisticated and resourceful. A wide spectrum of cutting-edge machine learning techniques, including deep learning and big data analytics, have been proposed for a variety of applications. New benchmark NIDS datasets have been jointly developed and evaluated [172]. In general, today we observe a closer collaboration between the AI and security communities. In light of these new developments, we revisit the primary conceptual and practical problems of ML-based NIDS.

5.4.1 Original Premise

The underlying assumptions of machine learning underpin (open-world) ML-based NIDS solutions. For open-world unsupervised or semi-supervised network intrusion detection to be effective, the following assumptions have to hold:

- *Benign data assumption* – there exists a region with well-defined boundaries that encompasses all the normal traffic data.

- *Clean training data* (for semi-supervised approaches) – benign training data acquired by collecting live background traffic is attack-free.
- *All attacks are rare and anomalous* – traffic generated by malicious actions related to network intrusions deviates sufficiently from the normal traffic and will only constitute a small fraction of monitored data.
- *All anomalies are malicious* – whenever a deviant pattern is observed, it presents evidence of a potential intrusion.
- *Attacks are universal* – given correct modeling of normal data, all types of attacks are detected equally well. In case of supervised learning, this assumption can be rephrased as: given enough labeled malicious data, a trained model can target all known attacks at once.

To enable the full potential of ML, these assumptions have to align with domain-specific characteristics, which in the case of securing dynamic and modern network environments is not a trivial question. Naturally, the extent to which these properties can be safely assumed differs from one environment to another and strongly depends on the threat model of a NIDS. The community is attempting to assess these fundamental assumptions behind all types of ML-based NIDS:

Can normal/benign data be accurately modeled? Most of the studies attempt to model benign traffic; however, not all benign behaviors follow a common distribution. It is overwhelmingly hard to completely capture the notion of "normality", so the safest assumption to make is that the model cannot describe all possible benign instances. This means that false alerts and missed attacks are unavoidable, and adjustment to novel benign patterns is necessary, which we discuss further in the section.

Is it possible to acquire clean training data? The current consensus is that normal traffic collected in a live environment is never attack-free without additional (manual) sanitization.

Are attacks rare? Certain illegitimate activities in the network (e.g., scanning) have become so common that they comprise a large fraction of background traffic [232]. Durumeric et al. [63] revealed that DDoS cannot be considered rare or anomalous in most networks. This needs ot be taken into account when modeling the correct ratio of benign and malicious traffic patterns in the wild. Moreover, this prevalence of large-scale attack traffic only further exacerbates the challenge of obtaining attack-free data for training. However, even though large-scale attacks are not rare, these are not of the biggest interest for detection. More sophisticated intrusions such as APTs are still manifested in rare events.

Are attacks anomalous? The answer directly relates to the vague definition of traffic "normality". Due to the noisy and highly varied nature of traffic, attack features may in practice appear as variations of benign traffic. Iglesias et al. [97] have recently conducted an analytical study to assess the "outlierness" of malicious traffic. They confirmed that network attacks have higher global distance-based outlierness averages; however, attack and normal traffic distributions strongly overlap. One can choose the feature space that maximizes the separation of benign and malicious traffic, which indicates that understanding the nature of target anomalies in a certain scenario is instrumental for anomaly detection. Another known issue is that attackers may attempt to make traffic features indistinguishable from normal traffic. We elaborate on the associated risks further in the section.

Are attacks universal and equally detectable? Taking everything into account, there is little ground in assuming that different types of intrusions can be detected in one common manner. Moreover, the very definition of what is malicious differs across environments. Indeed, we observe the trend of developing NIDSs tailored to specific threat models. This includes, e.g., works that focus on botnet detection [177, 113], DDoS detection [200], and especially APT detection [127], where data exfiltration through the network can be a target anomaly. It is quite unlikely that such targeted detectors generalize to other types of intrusions, but perhaps that should not be the initial goal. ML algorithms may excel when they focus on specific threats instead of trying to detect all threats at once. We advocate for deeper insight in targeted malicious activities even for open-world anomaly detectors, in order to adopt the most suitable strategies.

Does an alert mean there is an attack? Nowadays, it is commonly acknowledged that an attack detected by a NIDS is most probably a false alert. Even correctly detected anomalous patterns are not always of a malicious nature: sometimes, deviations happen due to noise, changes in the underlying infrastructure or in the benign data distribution. Therefore, additional processing is required to investigate the alert, as monitoring and detection are just the earliest stage in the complex process of incident management. Additional analysis, attack correlation and response planning is a prerogative of SIEM platforms [25]. While researchers have mainly focused on developing effective solutions for detection, studies on automatic intrusion response are still limited. The main challenge is in providing an accurate and informative description of the detected attacks, including *interpretation* of the ML model's decision to raise an alert. Further on, we elaborate on minimizing the rate and impact of false alerts.

5.4.2 High Error Rates and Performance Estimation

If one has to highlight the main problem with the adoption of machine learning in mainstream network security systems, high False Positive Rate (FPR) is an immediate candidate. For an enterprise IDS, manual investigation and interpretation of alerts consumes expensive analyst time. Given the large volumes of processed data and a low *base rate* of attacks of interest, even a very small fraction of false alerts generated by a nearly-perfect model yields an unacceptably large absolute number, effectively rendering a NIDS unusable in the operational setting. This issue of *base rate fallacy* was raised two decades ago [15], and is seen today as an inevitable pitfall of open-world detection: *precision of an IDS will always be determined by both the base rate of different attacks and the FPR*. Regretfully, however, we lack historical statistics for the base rates of attacks in real computer infrastructures, and measuring them reliably is still considered beyond present capabilities [153].

As the tolerance for errors in the application domain is critically low, researchers started advocating for placing more emphasis on constraining the FPR while preserving high detection rates [196]. Since then, more studies have targeted this specific problem. We observe that the solution space can be mainly branched into five complementary directions: (i) further developing more precise learning algorithms to lower the FPR [4, 69]; (ii) post-processing alerts with the use of context or prior knowledge in the system [235, 80], in order to aid in manual diagnostics and potentially understand the nature of an anomaly; (iii) employing a *hybrid* approach by combining anomaly detectors with misuse detectors [81], which cannot detect novel attacks but are considered less prone to mispredictions; (iv) tuning model parameters and detection thresholds in order to obtain optimal trade-offs in success rates and false alerts [197]; and (v) modeling a realistic network environment in a structured manner and choosing appropriate metrics to correctly estimate the FPR and the overall performance.

While the first four objectives are gradually unfolding in present research, the last one is fundamental and largely remains an open question. It relates to the inherent difficulties with evaluating an open-world detector, which started being actively discussed more than ten years ago [76, 74, 196] and still hold today. With more progress in this direction, future NIDS studies should adopt an appropriate evaluation methodology and correct metrics that correspond to an actual operational usage of the target system. This requirement encapsulates such a crucial issue as validating and testing the model on data that resembles real-world ratios of benign vs. attack data – which again relates to the base rate fallacy. Without satisfying these goals, performance numbers and errors rates achieved in lab conditions will remain hardly reliable or comparable. The issue is especially pronounced for unsupervised methods, which learn from distributions

and spaces drawn from the observed data. Note that even modern benchmark datasets are not said to be representative of an actual ratio of normal and attack traffic, therefore they are most often not directly applicable for realistic evaluation schemes. We further elaborate on the representativeness of existing datasets below.

In summary, the research on decreasing the FPR while preserving performance in a general NIDS setting is still unfolding. Despite some studies emphasizing the post-processing stage of predictions, there is generally not enough investigation being made into the nature of false alerts, while most of the works solely focus on increasing the detection rates instead. Even though NIDSs with manageable error rates are allegedly becoming more widely adopted in industry, these solutions are often designed for specific scenarios and their internals are rarely publicly available [74, 4, 84], preventing direct comparison. The field appears to be in urgent need of a common comprehensive methodology for estimating and comparing performance and error rates of a NIDS.

5.4.3 Representative Datasets and Ground Truth

In IDS research, evaluation on benchmark datasets primarily serves a two-fold purpose: (i) *real-world performance estimation* of a particular algorithm, and (ii) *consistent comparison* between different approaches. In this respect, quality of data has a decisive influence on the valid outcomes of both objectives. Unfortunately, benchmark datasets do not always adequately represent the real problem of network intrusion detection, discrediting the performance numbers achieved in laboratory conditions. As a response, over the last 10 years the community has collectively devised criteria that reliable research traffic data should meet [207, 147, 188, 217, 186, 65], which encapsulate the following dataset properties: (i) realistic w.r.t. real production environments; (ii) valid w.r.t. completeness of traces; (iii) labeled; (iv) correctly labeled w.r.t. benign training data for anomaly-detection; (v) highly variant and diverse w.r.t. used services, protocols, benign behaviors and attacks; (vi) correctly implemented w.r.t. real attack scenarios; (vii) easily updatable with new services and attacks; (viii) reproducible for periodical updates and performance comparisons; (ix) shareable/non-sensitive; and (x) well-documented. Despite this recently achieved consensus and clarity in guidelines, a lot of fundamental limitations of the task hamper both creation and publication of a corresponding proper dataset. Consequentially, many researchers have kept using the existing suboptimal datasets for the sake of comparison with prior work. Unfortunately, the usage of outdated or even novel but flawed datasets may lead to unreliable ML evaluations, as has been recently shown by Engelen et al. [65] and Liu et al [119]. Nevertheless, the research community is making tangible progress in

contributing more representative datasets by either generating synthetic traffic, or collecting real traffic in a production environment.

Generation of synthetic datasets provides the luxury of a controlled environment, clean labels and no privacy concerns. The main challenge, however, is in simulation of realistic background traffic, lately attempted through statistically modeling user behavior [188, 185] or more recently through modeling relevant traffic communication scenarios with input randomization [46, 45]. Even though creators of modern synthetic datasets strive to minimize occurrence of *simulation artifacts*, the practice of evaluating a novel IDS on a synthetic dataset solely is often criticized as insufficient. While it can be reasonable to compare different frameworks on synthetic data, evaluation on diverse network traffic collected in a live environment over a lengthy period of time is becoming the desired norm in NIDS research.

Real traffic, on the other hand, should be stripped of confidential data, carefully labeled and rigorously sanitized in order to meet the established criteria. Several studies contributed approaches to sanitization of traffic [23, 48, 82, 216] in order to not only label embedded attacks and benign traces, but also to pre-select the most representative instances. Automated sanitization uses such methods as entropy analysis and signature-based attack labeling, which may result in erroneous ground-truth. Manual sanitization hardly scales and is prone to human bias, which threatens reliability and representativeness of the dataset, respectively. However, manual supervision in labeling seems unavoidable when it comes to zero-day network attacks.

In summary, it is unclear whether a perfectly sanitized real traffic-based dataset can be obtained. Hence, learning algorithms that are robust to the inevitably occurring noise in labels would give a strong advantage from an operational point of view. Promising examples for anomaly detection on imperfectly labeled traffic include robust PCA algorithms [157, 128] and a convex combination of anomaly detectors' outputs [79].

Another suggestion for creation of an open NIDS dataset with real network traffic was voiced by Gates et al. [76], who promoted a community-based approach. One prominent example is the MAWILab dataset [71] – a public repository for automated labeling and performance estimation that has since been continuously updated and collectively labeled with the use of state-of-the-art anomaly detectors. While anomaly detection solutions on these data are still scarce [35], we believe that such collective efforts establish a strong foundation for open-world detection research.

5.4.4 Concept Drift

In dynamic environments, events undergo gradual and abrupt changes over time, which causes a shift in data distribution known as the *concept drift* [162]. When developing data-driven and real-time defensive solutions such as ML-based IDSs, it is crucial to account for concept drift, otherwise the model's performance is unpredictably and heavily impacted. For an anomaly detector, this implies the need to track drift in the data in order to continuously adjust to the new definition of normal behavior, instead of erroneously flagging these changes as anomalies. A direct way to re-adapt the system accordingly is to re-train the model on new data, as is strongly recommended in the literature [196, 74, 132]. In anomaly detection literature, the problem of detecting newly emerging patterns is referred to as *novelty detection*, when previously unobserved detected patterns in data are incorporated into the normal model. In time-series analysis, a similar idea is defined as *change point detection* [19] that aims to detect points in a time-series from which the data distribution changes. Conventional approaches often suggested for ML-based defenses aim to detect concept drift by recognizing the model's performance degradation on streaming data and identifying the appropriate moment for a model update. Naturally, the crucial trade-off emerges between the detection delay and the detection quality. Some families of ML algorithms such as neural networks can adapt through a continuous retraining mechanism known as *online/incremental learning*, exemplified by a DL-based IDS that analyzes log data [62]. By incorporating the most recent changes in system logs into the DNN model, a DL-based IDS can adjust to newly emerging patterns in a timely manner. Incremental learning can also be applied to traditional ML algorithms, albeit with high computational complexity. For instance, Rassam et al. [165] utilize an adaptive principal component classifier-based anomaly detector that tracks dynamic normal changes in real sensor data. However, effectiveness and practicality of incremental learning or re-training for an anomaly IDS on network traffic – non-stationary streaming data – largely remain unexplored. Raza et al. [166] developed a theoretical approach that addresses detection of covariate shifts in generic non-stationary environments and can potentially aid in IDS concept drift. One promising approach to an autonomic anomaly NIDS was proposed by Wang et al. [227], who use novel clustering algorithms to label new data and dynamically adapt to normal behavior. They show efficacy of their algorithm on a private dataset of real HTTP traffic streams. Zhang et al. [239] employ a competing approach specifically tailored to high-dimensional streaming data. To account for concept drift, they perform adaptive subspace analysis that fully relies on human feedback to prune away irrelevant subspaces of anomalies. As this novel algorithm is only evaluated on the KDD'99 dataset, its ability to generalize to real traffic and scale to live environments is unknown. Dong et al. [58] developed a batch-based

adaptation approach that utilizes an SVM classifier and incorporates human feedback to determine when re-training is necessary. Their evaluation is limited to malicious web requests, and they use a public dataset with HTTP traffic.

A recent advance in the space of concept drift detection and adaptation techniques for NIDSs is INSOMNIA by Andresini et al. [10]. They deploy a semi-supervised intrusion detector which continuously updates the underlying deep neural network classifier as network traffic characteristics change with the concept drift. Their evaluation focuses on classification (closed-world binary classification), but demonstrates successful adaptation to drifting scenarios in a simulated benchmark dataset.

A large roadblock for research on concept drift is the lack of public representative benchmark datasets that contain labeled shifts in traffic. A notable recent contribution is the UGR'16 dataset [123] – real anonymized Netflow data for adaptive NIDS research that includes long-term traffic evolution and periodicity.

5.4.5 Real-Time Detection

In the era of growing risk and severity of cyber-attacks, an effective NIDS is expected to detect potential threats immediately as they occur in the network. An ideal *real-time* detector processes and analyzes a continuous stream of data in its natural sequential form and makes immediate decisions online [138]. Anomaly detection is regarded as indispensable in early open-world detection of novel, unusual behaviors, and yet the existing approaches are not effective enough in real-time detection [4] and still largely resort to offline analysis, or batch processing at best, allowing some intrusions to go unnoticed for days. In the meantime, the bar for real-time processing capabilities is only increasing: not only does the internet traffic double each year, but in addition to that, the growth of the Internet of Things (IoT), sensors, smart cities, mobile clouds, autonomous vehicles, and other emerging technologies has unleashed enormous amounts of generated network data. Cisco has reported [72] that by 2022, the omnipresent non-PC devices are estimated to drive 81% of global internet traffic, opening the gate for more large-scale network attacks against small connected devices. The network data of today is already characterized by huge volume, velocity, variety and veracity, fulfilling the definition of *big data*. Traditional ML-enabled NIDS have not been developed to handle big data, but largely aimed at enhancing learning algorithms, which mostly results in increasing the computational complexity and processing time [4, 31, 69], further hindering real-time analysis.

As the demand for effective security monitoring raises higher by day, novel solutions are required to facilitate large-scale, real-time detection. Hoplaros et

al. [95] explored *data summarization* techniques that mine patterns in summaries of network traffic to approximate final decisions and improve efficiency of detection. Since this approach effectively allows to cut offline detection runtimes, the authors propose to develop stream data summarization and distributed summarization methods for online detection. The downside is that complex summarization on big data contributes to opacity of model predictions, while threat analysis benefits from more granularity and transparency in decision-making.

Collaborative intrusion detection systems that employ several distributed monitors for collection and analysis of traffic pose an alternative to the bottleneck stand-alone anomaly detectors. A collaborative NIDS is considered to be much more efficient in analysis of numerous data streams traversing through large networks and IT ecosystems. Vasilomanolakis et al. [213] provide a taxonomy and a detailed survey on the topic, including possible topologies and threat models for a collaborative NIDS. Zarpelao et al. [236] presented a survey of stand-alone and collaborative IDS solutions specifically for IoT infrastructures. Crucially, most of the modern NIDS research on large-scale networks, IoT in particular, does not provide enough details for reproducibility and use private specifically chosen testbeds or simulation tools. Moreover, the internal mechanisms of existing commercial products are also hardly available. In summary, a thorough investigation on public data with a standardized evaluation strategy is required to assess effectiveness of a collaborative NIDS in real-time detection of sophisticated attacks in modern network environments.

Suthaharan et al. [206] were among the first to highlight the challenging big data properties associated with network monitoring for security. They advocated incorporating known big data frameworks, e.g., Hadoop [191], into a ML-based NIDS framework, in order to combine *big data processing* tailored to real-time analytics with supervised ML classifiers and representation-learning techniques. This integration requires to rethink implementations of ML algorithms in general and introduce *parallelization* by either dividing data into separately processed subsets, or dividing a ML algorithm into concurrently performed steps. Later on, the discussion was extended to anomaly-based NIDS in order to enable real-time open-world detection on large streaming data [68, 134, 133, 164, 35]. Recently, Habeeb et al. [84] have thoroughly reviewed real-time network anomaly detection algorithms and discussed the aspects and challenges of their application to big network traffic data. Despite the promise of big data frameworks widely deployed in other domains, in network security we still observe a premature state of big data processing capabilities. Efficient model and parameter selection, automation of data filtering and curation, dynamic resource allocation, reduction of power and memory consumption are only a few associated future research directions. With these enhancements,

advance anomaly detection in combination with modern big data tools should be adequate to handle large-scale real-time detection, feature extraction and selection, labeling, and model retraining.

5.4.6 Adversarial Robustness

Attackers have always had great incentives and tools to evade detection by a NIDS. Knowing the blind spots of a detector, an adaptive attacker chooses the optimal strategy that fools a NIDS into thinking that the passing traffic is legitimate. Misuse-based detectors that inspect traffic on the packet level are traditionally evaded through such means as encryption, obfuscation and packet fragmentation, which make sure that malicious traffic does not match a known signature. A general anomaly-based detector is vulnerable to *mimicry attacks*, which modify malicious traffic in such a way that it corresponds to normal traffic patterns [107, 70]. Besides, any type of a NIDS is susceptible to various DoS attacks, which can overload the detector with meaningless connections to create a bottleneck, so that the actual malicious connection comes through unnoticed. Security analysis of novel NIDS solutions with respect to adaptive attackers is regarded as a crucial research angle.

In the last decade, more attention was brought specifically to evasion of ML-enabled defenses. When placing a ML model at the core of a defensive system, one will always involuntarily introduce a new attack vector of undetermined severity. *Adversarial machine learning* is a set of techniques that exploit specific vulnerabilities of ML algorithms in order to trigger an incorrect output. These vulnerabilities arise from the same aspect that makes ML models unadapted to concept drift: ML models that are initially designed and evaluated in a stationary environment without any external influences assume a certain data distribution with static priors. The guarantees provided by this evaluation are only valid for the expected inputs to the model, i.e., the datapoints generated from the same data distribution that was considered during training and testing. NIDSs, however, are deployed in *non-stationary adversarial environments*, where attackers may choose to purposefully alter and perturb the input data to force the ML model to fail, thus inducing *adversarial drift* [106]. Adaptive attacks that specifically target vulnerabilities and assumptions of ML algorithms are referred to in the literature as *adversarial attacks*.

Corona et al. [47] describe a general adaptive threat model for a NIDS and review studies of one particular category of adversarial attacks, namely *poisoning* attacks. In case of ML models that are designed to automatically adapt to changes in normal traffic, either through online learning or retraining, an attacker can poison the model by inserting adversarial noise in seemingly benign

traffic that is consequently used for training. After compromising the learning process in this way, the attacker can bypass the poisoned model by manipulating input samples in accordance with the new decision boundary. Essentially, as a result of a poisoning attack, an attacker establishes a *backdoor* in the ML model that allows to evade detection. For instance, Kloft and Laskov [111] explored poisoning attacks against centroid anomaly detection and confirmed its effectiveness on feature vectors representing real HTTP traffic.

The second, most known type of adversarial attacks is *evasion*, where an attacker constructs a malicious *adversarial example* by introducing carefully crafted perturbations in input traffic that cause a desired output of the target model while preserving the validity and other desired properties of the input. In case of a targeted attack, an adversary attempts to influence the exact outcome of prediction at inference time. Meanwhile, an untargeted attack succeeds as long as any misprediction occurs. To construct adversarial examples that effectively evade detection, an adversary should either have direct access to some of the target's model properties (architecture, parameters, training data, input features, etc.), or simply be able to continuously query the model as an oracle and analyze its output. These threat models are referred to as *white-box* and *black-box* respectively, with the latter assuming minimal knowledge about the target NIDS. While the black-box scenario is more appropriate in network security, white-box approaches to generation of adversarial examples are also applicable assuming a local surrogate ML model whose decision boundary is close to that of the target.

Although all these approaches are algorithmically feasible in terms of mathematical formulations, their practicality and impact on security of real network environments are just starting to be comprehensively researched. Preliminary research results on closed-world systems (e.g., evasion of botnet classifiers trained on Netflow features [11]) indicate that ML algorithms used behind NIDSs are indeed not robust to perturbations. However, at this point, it remains unclear whether adversarial attacks against open-world network intrusion detection systems can withstand domain-specific practical constraints. One of the obstacles to investigating this question is the absence of dedicated research datasets that include sophisticated ML-specific evasion attempts in raw network traffic. Another more fundamental concern is the overall utility of adversarial learning in this context, that is how compelling this approach would be for a network attacker. While introducing ML into a NIDS pipeline undeniably creates additional tangible threats, other more conventional attack vectors – such as the above-mentioned mimicry attacks or DoS attacks that overload the detector – appear much less costly or complex to perform than adversarial attacks against ML. As the practicality and impact of adversarial attacks in the operational scenario of a NIDS remain unknown, and other

challenges in using ML effectively keep affecting operational performance of NIDSs, hardening detectors used in actual services and infrastructures against adversarial perturbations has not yet gained the envisioned priority.

5.5 Conclusion

Machine learning for network intrusion detection is an especially intriguing and potent research direction, which, despite its strong theoretical base, seems to lack devoted attention in defensive security applications. Today, the community recognizes the non-stationarity and adversarial nature of security applications, promoting thoughtful and realistic evaluation of effective and adaptable ML-based defenses. For this, hard truths were encountered about domain-specific properties and limitations of ML in open dynamic environments. While one cannot create a silver-bullet solution to network intrusion detection, we can deepen our understanding of the underlying issues and provide fundamentally sound ML techniques for NIDS.

This chapter reviewed the wide spectrum of impressive research efforts in the area of anomaly-based NIDS and highlights the main challenges to become the focus of future research. Our analysis encompassed domain misalignment with the original assumptions of anomaly detection, high error rates, the problem of performance estimation and comparison, availability of realistic datasets and reliable ground truth, adaptability to concept drift, feasibility of real-time detection, and adversarial impact. From our discussion, it becomes evident that there is no clear-cut separation between various challenges or desired properties of (open-world) detection systems. Moreover, the undeniable benefits of using ML in a security context come together with a number of concerns and even incompatibilities between the techniques and the problem at hand. Hence, a careful balance between entrusting security tasks to ML and yet being aware of its potential pitfalls may bring out the most reliable and fruitful solutions.

Therefore, in order to progress the field, the future research is bound to reason about network intrusion detection in a more principled way that considers all important aspects in conjunction and allows to systematically assess how they affect each other. To achieve that, we encourage the community to collectively devise appropriate ML methodologies to develop and evaluate realistic open-world network intrusion detectors in different environments and threat models. As this is a tough task for the years to come, we need to scale the research by composing benchmarking scenarios under a common set of assumptions to compare novel methods in a fair and informative way. To this end, coordinated

efforts and an open science approach to NIDS research, including new open realistic datasets and open-source implementations, deserve the highest priority.

An attentive reader has most probably noticed that the discussion of explainability of ML-based NIDS has not been as expansive as discussions of other key aspects. We elaborate on explaining and interpreting ML-based NIDS in the context of deep learning for network intrusion detection in the next chapter.

Chapter 6

Deep Learning for Network Intrusion Detection

> "It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts."
>
> — *Arthur Conan Doyle (A Scandal in Bohemia, 1892)*

6.1 Preamble

This chapter elaborates on our ongoing exploration of deep learning in the context of network intrusion detection, focusing on representation learning from raw sequential traffic representations. Designing defensive deep neural networks for Network Intrusion Detection Systems (NIDSs) presents several major challenges. This chapter reports on our research efforts to expose and understand the failures of deep neural networks due to shortcut learning. We describe a practical, systematic approach that we envision integrating into commonly used evaluation strategies to perform proactive failure analysis of deep learning-based NIDSs.

The centerfold of this chapter is a methodology to study, reveal, and visualize shortcuts, i.e., spurious correlations and overemphasized features in data, that automatic feature extraction may succumb to in the absence of manual guidance by a human analyst. In our analysis, we recognize the impact of various

architectures, traffic representations, data collection aspects, and the considered attack scenarios on vulnerability to shortcuts, thereby motivating a focused future analysis of each variable. The chapter concludes with encouraging preliminary empirical results, which present a bridge to future work on systematically overcoming the inherent pitfalls of deep neural networks as a defensive solution in network security. This contribution is a step towards assessing and ensuring valid, intended behavior of data-driven defenses in network security.

6.2 Introduction

A NIDS plays an integral role in securing modern network environments. As the sophistication and stealth of network-based threats increase, traditional rule-based and probabilistic data-driven Machine Learning (ML) methods can be utilized to detect and respond to known and earlier unknown attacks. A NIDS can recognize a known attack by its template – this is so-called misuse detection or signature-based attack recognition. Alternatively, a NIDS can be designed to detect suspicious network activity when its characteristics are sufficiently different from the expected normal behavior – the network anomaly detection approach.

One needs to determine the appropriate traffic format that a NIDS can interpret and analyze to perform traffic analysis. Payload inspection has proven to be an effective approach to NIDSs which aim to detect malicious activity from the content of network packets [112, 126, 220, 219]. Nevertheless, even the most efficient packet inspection solutions that avoid high computational overhead face several challenges. Increasing concerns for the confidentiality of internal and external communication, and public awareness about individual privacy, are behind the demand for ubiquitous end-to-end encryption of network data. As encrypted traffic becomes increasingly prevalent through the mass migration to secure network protocols, payload inspection becomes less applicable and, in certain cases, even prohibited by law and privacy policies. As a result, a modern NIDS cannot always rely on the availability of payloads and may have to utilize supplementary ways.

Therefore, detection methods commonly rely on traffic metadata. Traffic metadata analysis is a generally applicable approach to intrusion detection that does not rely on packet contents but on patterns of network flows [240, 156, 87, 36]. A flow-based NIDS operates on metadata of the communication, including the timing and sizes of network packets, enabling it to handle both encrypted and unencrypted traffic in the same manner. Traditionally, traffic analysis experts

use these metadata to manually define statistical characteristics of network flows that are useful for detecting known malicious or unknown abnormal network activities. Irrespective of the choice of the ML method, Random Forest (RF), Support Vector Machine (SVM), K-Nearest Neighbour (KNN), etc., the performance of a NIDS depends heavily on the quality of the manually chosen traffic representation. However, hand-engineered features are limited in terms of generalizability and might decrease in relevance over time, prompting continuous manual updates following evolving traffic patterns or adversarial activity. To handle these challenges, security analysts and engineers have been continuously upgrading statistical features that tell most of the benign or malicious nature of observed traffic.

After the initial success of deep learning at other traffic analysis tasks [228, 9, 120], Deep Neural Networks (DNNs) started being considered for intrusion detection. In addition to eliminating human bias in feature engineering, DNNs can automatically find more complex implicit features within the input data through many layers of non-linear transformations. The first uses of deep neural networks for a NIDS analyzed high-level traffic statistics, similar to the state-of-the-art ML algorithms. This approach, however, does not allow for assessing the potency of automated feature extraction since such a model already operates on manually composed traffic representations. In network security, automatically learning and readjusting rich traffic representations from raw input with DNNs is a compelling operation, as it could create viable defenses against constantly evolving and newly emerging network attacks [234, 241, 179, 135, 129].

Other open questions concern the impact of deep neural networks on the operational side of network intrusion detection and the trustworthiness of made predictions. In the defensive context, erroneously flagged benign activities and undetected attacks pose serious threats to the protected system. In order to diminish potential recovery costs and prevent a disruption of services, security analysts are interested in somehow verifying that a deployed non-interpretable DNN behaves as intended. The simplest but hardly scalable way is to manually check and troubleshoot erroneous automated predictions. In this regard, the black-box nature of DNNs may present a challenge, in contrast to the more interpretable combinations of hand-crafted statistical features and shallow ML algorithms.

In summary, it is unclear whether deep learning can realize practical network intrusion detection in raw traffic and, most importantly, whether it provides benefits that would justify the added complexity of DNNs and their non-interpretable learned representations.

Research goals. In this study, we aim to contribute to answering the following question: *Is automated feature extraction with deep neural networks a viable component of flow-based network intrusion detection?*

Specifically, our research on a deep learning-enabled NIDS holds a twofold objective: (i) an application of automated feature extraction on raw traffic and (ii) a critical analysis of this approach in terms of its advantages and disadvantages for the domain. Our study aims at addressing these objectives through the following two stages.

Network Intrusion Detection through Sequential Feature Learning

In the initial stage, we assess the applicability of the automated feature learning approach compared to traditional ML on statistical features. We propose a NIDS processing and learning pipeline tailored to learning from sequential multi-dimensional data, i.e., raw traffic flow representations. Our solution utilizes convolutional and temporal neural networks to implement sequential feature learning from stacked packet headers in monitored traffic flows. While doing so, the sequential NIDS obviates task-specific or class-specific manual feature engineering and costly payload inspection by design. Concurrently to this stage of the project, an alternative deep learning-based design was proposed in the literature, named LUCID [59], with a focus on resource-constrained network environments. Both solutions are evaluated on benchmark datasets in application to misuse detection: i.e., distinguishing known attack patterns from benign traffic and between different attack categories. Deep neural networks perform better than traditional ML algorithms that operate on high-level statistical flow features.

However, experimental results on a benchmark dataset are only the initial component of estimating the efficacy of NIDSs in the case of real-world deployment. In order to validate the utility of deep learning-based approaches, an appropriate evaluation methodology is required. The primary desired properties a model should provide in order to be used for network intrusion detection are (i) a valid underlying logic for the problem at hand and (ii) generalization capabilities beyond the benchmark dataset used to develop the model. These properties cannot be addressed without realizing at least the following two fundamental aspects: utilizing valid and representative data with a suitable level of abstraction and empirical verification of the intended behavior of a NIDS both within and beyond benchmark data.

Hence, for this study stage, we must ensure high-quality data for evaluation. For that, we build on our previous analysis [65] of a popular modern dataset with synthetic traffic that consists of two required traffic formats: the original traffic

captures and the corresponding statistical features used in the literature. The result of this previously published analysis is a correction of flaws in the original dataset related to valid flow construction, feature extraction, and labeling. The final dataset is made publicly available for future research and is continuously updated.[1] Following this effort, Liu et al. [119] updated another large and more recent dataset. Their work reaches similar conclusions and further underlines the necessity of high-quality data in NIDS research.

Failures of Neural Networks on Raw Traffic Representations

As previously briefly outlined, the second stage of the study addresses the issue of verifying the intended behavior of deep learning-based NIDSs in practice. Due to the limited explainability and interpretability of DNNs, it is not trivial to arrive at a thorough scientific understanding of how deep learning-enabled defenses work. A potentially dangerous consequence can be unexpected failures of the deep learning-based NIDS in real environments, despite the highest performance demonstrated in lab conditions. This work is motivated by the demonstrated failure of the evaluated sequential NIDSs, both our own and the alternative designs from the literature, on known attack patterns with minor practical semantic modifications. This effect puts in doubt the generalization abilities of deep learning-based NIDSs and underlines the importance of its critical evaluation. The hypothesized reasons behind the failures are as follows. On the one hand, neural networks may be especially sensitive to artifacts in data, which occur due to issues with synthetically generated datasets. If the artifacts in data are present in both the training and test design stages of a NIDS, a model will learn erroneous patterns and fail on novel data. On the other hand, neural networks suffer from simplicity bias: the model might overemphasize certain features and downplay other salient features, which leads to weak generalization properties. Emphasizing only a few features may occur when the dataset used for network intrusion detection is not sufficiently diverse to represent real-world data. Together these two issues are the instances of *shortcut learning* [77] – a phenomenon recently demonstrated in other deep learning applications, within and outside the security domain. While shortcut learning seems to be caused by issues with the underlying data the DNN is trained and tested on, one could argue that data will always be imperfect. Therefore, understanding how different learning algorithms behave when applied to imperfect data and which other factors influence their vulnerability to shortcut learning is important for proposing trustworthy defenses. Specifically, evaluations are driven by the common Independent Identical Distribution (IID) about the data, which causes

[1] `https://intrusion-detection.distrinet-research.be/WTMC2021/`

poor generalization on Out Of Distribution (OOD) samples, where features shift to unpredictable distributions.

Contributions. In summary, our contributions are as follows:

- We overview deep learning algorithms applied to flow-based network intrusion detection in raw traffic representations, as opposed to hand-engineered high-level statistical features. To that end, we explore various DNN architectures, including temporal and recurrent neural networks and encoder-decoder architectures for an open-world design. Our analysis also includes an alternative design from the literature [59], which appeared concurrently with our study.
- We provide a critical analysis of performing automated feature extraction from various raw traffic representations, focusing on its limitations and unintended consequences. We show how, in the context of modern, not artifact-free NIDS datasets, neural networks can achieve misleadingly high results while the flawed underlying patterns detected by the model remain undetected.
- To mitigate the inherent non-interpretability of the generated predictions, we propose a methodology to systematically expose and analyze artifact-triggered shortcuts learned by a NIDS model. Our methodology builds upon state-of-the-art explainability algorithms and suggests domain-specific heuristics to guide the search for shortcuts. This contribution is a stepping stone towards establishing standardized and practically accessible error analysis for NIDS research.
- Our study highlights scenarios where our framework allows us to pinpoint a suspected shortcut, investigate its possible causes, and assess its prevalence in the dataset.
- We intend to publish the visualization framework to facilitate a joint exploration of error analysis on NIDSs that work with neural networks and non-interpretable traffic data.

6.3 Network Intrusion Detection through Sequential Feature Learning

In this section, we present the dataset, data representations, deep learning architectures and evaluation results of sequential feature learning for NIDSs,

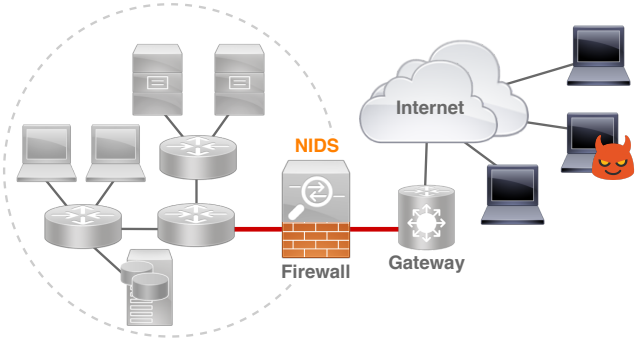


Figure 6.1: Network intrusion detection threat model considered in this chapter. A NIDS is deployed at the border, monitoring relevant network segments (in red) of the victim network, observing incoming and outgoing traffic.

including our own proposals and concurrent designs from the literature. We discuss how deep learning architectures proposed for network intrusion detection compare to other ML-based NIDSs and their corresponding features. The models and representations considered in this part of the study further serve as the target NIDS approaches explored in terms of their robustness to shortcut learning.

6.3.1 CIC-IDS-2017 Dataset

The CIC-IDS-2017 dataset is a NIDS dataset that was created by the Canadian Institute of Cybersecurity (CIC) in collaboration with the University of New Brunswick. The dataset was collected in a testbed environment, and contains benign traffic with simulated benign profiles as well as malicious traffic from 14 different attacks. The attack scenarios used in the dataset simulate network intrusions from external adversaries against a victim network, as is depicted in Figure 6.1. This threat model is used throughout this chapter.

Corrected Labeling. In our previous study [65], during a manual exploration of the PCAPs at the base of this dataset, we discovered that over 20 percent of all segmented traffic traces were artifacts resulting from a mistake in the feature extraction tool and inherently do not contain any meaningful information. Moreover, these artifacts were semantically identical across all classes. Yet, a Random Forest classifier was able to "correctly" label these artifacts according to their ground-truth in the dataset. This strongly indicates that the classifier was relying on shortcuts for "correct" classification of these artifacts. Our previous

study contributes a corrected version of CIC-IDS-2017. The main issues this corrected dataset resolves are the flow construction (flaws in which were the cause of the appearance of meaningless artifact flows) and the incorrect ground truth, which was corrected by manually revising the labeling logic. We use the corrected version of the dataset for benchmarking the models and comparing to traditional ML.

Data Representation. The creators of CIC-IDS-2017 also include the original raw traffic, making it possible to map feature vectors back to corresponding initial network flows. Since the PCAP files are available, by modifying and re-purposing the fixed CICFlowMeter tool we extract the sequences of packets within each flow, which resulted in labeled variable length packet sequences. First we omit the payloads and transform packet headers directly to sequences of numerical header attributes, resulting in a set of packets $P = \{p_i, ..., p_N\}$ with each p_i represented as a 12-dimensional header attribute vector h, listed in Table 6.1. P can be divided in separate labeled traffic flows $F = \{f_i, ..., f_M\}$ with $M < N$ where each flow is defined as a host-to-host connection identified by a unique combination of source and destination IP addresses, port numbers and transport protocol. Each flow is a subset of header attribute vectors of temporally ordered packets in P transmitted between two hosts in both directions (with interchangeable IP addresses and ports). Each flow segment used as an instance for learning and inference is a part of a flow corresponding to one network connection, and is labeled as benign or malicious.

Table 6.1: Connection-based representation: attributes at each timestamp of a traffic flow sequence.

#	Attribute	#	Attribute
1	Relative Timestamp	7	TCP.ACK
2	Direction	8	TCP.PSH
3	Destination Port	9	TCP.RST
4	IPv4 Total Length	10	TCP.SYN
5	IPv4 No Fragment	11	TCP.FIN
6	TCP.URG	12	TCP.Window

LUCID proposes an alternative representation, where the flows are segmented not based on a network connection, but based on a time window. Therefore, one segment might have parts of different connections. The features that are evaluated in the study also differ and are more abundant. The LUCID representation is applicable to all transport protocols, rather than just TCP. We

list them in Table 6.2. Note that LUCID converts the protocols from a binary string to real numbers: "Binary representation of the list of protocols recognized in the packet using the well-known Bag-of-Words (BoW) model. It includes protocols from Layer 2 (`arp`) to common clear text application layer protocols such as `http`, `telnet`, `ftp` and `dns`" [59]. IP and TCP flags are hexadecimal values from Wireshark converted to decimal values.

Table 6.2: Time window-based representation (LUCID): attributes at each timestamp of a traffic flow sequence.

#	Attribute	#	Attribute
1	Relative Timestamp	7	TCP Acknowledgment number
2	Packet Length	8	TCP Flags
3	Highest Layer	9	TCP Window Size
4	IP Flags	10	UDP Length
5	Protocols	11	ICMP Type
6	TCP length	–	–

6.3.2 Deep Learning Models

To perform automated sequential feature learning together with inference on time-series, we adapt convolutional and recurrent neural networks (the definitions and more details on the working mechanisms behind these architectures can be found in Chapter 2). These algorithms provide the learning capability and architectural properties to learn on and interpret temporal dependencies in the available network traffic information.

There are two main evaluation setups for ML-based NIDS: open- and closed world detection. For open-world detection we reconfigured One-Dimensional Convolutional Neural Network (1D-CNN), Temporal Convolutional Network (TCN), Long Short Term Memory (LSTM) and Bidirectional Long Short-Term Memory (BiLSTM) for anomaly detection by placing them into a sequential encoder-decoder architecture, some with the use of the Attention mechanism [122, 214]. We use a naive setup where we train each anomaly detector on the benign class only (therefore, without using malicious data to learn the weights of the networks) and then perform binary classification on a balanced test set. Note that we include a minimal number of malicious samples in validation, to guide early stopping of the training process.

For closed-world detection, the following classifiers are designed and evaluated:

1D-CNN. Convolutional Neural Network (CNN) that applies learning kernels to input traces in one direction, along the packets (LUCID architecture [59]).

TCN. Type of a 1D-CNN with residual connections that has larger memory capacity than a classic CNN .

LSTM. Type of a Recurrent Neural Network (RNN) with increased memory capacity that analyzes a trace sequentially packet by packet.

BiLSTM. Type of an LSTM that processes the input sequence in both directions and merges the two results, e. g., through concatenation or summation.

For open-world detection, the main adaptation to the above models follows the fundamentals of (deep) one-class classification [108, 174] and an encoder-decoder architecture [17]. This allows to realize sequence-to-sequence modeling and perform anomaly detection by thresholding the reconstruction error of the decoder. The following deep autoencoders are designed and evaluated:

TCN AE. A TCN encoder and a time distributed [2] decoder.

CNN AE. A 1D CNN encoder and a time distributed decoder.

LSTM-Att AE. An LSTM autoencoder with attention [122, 214].

6.3.3 Evaluation and Results

This section reports evaluation results on the CIC-IDS-2017 dataset using the connection-based traffic representation. In Appendix C.1, we elaborate on the hyperparameter tuning process, including the search spaces and selected parameters.

Open-World Detection. The anonaly detection results on the CIC-IDS-2017 dataset with the connection-based raw representation are presented in Table 6.3. The anomaly detection models achieve a mean average precision (i.e., area under the Precision-Recall curve) of 91.3%, with the top performance of 94% by a TCN-based open-world NIDS. As a baseline, we use a shallow dense Autoencoder (AE) (i. e., a regular feedforward autoencoder with one hidden layer) on high-level statistical features. After extensive tuning, we reach 93% average precision with the baseline autoencoder.

[2] `https://keras.io/api/layers/recurrent_layers/time_distributed/`

Table 6.3: Performance comparison between anomaly detection models: average over 10 evaluation runs. A shallow dense AE is used as a baseline.

	Dense AE	**LSTM-Att AE**	**CNN AE**	**TCN AE**
AUC	0.87	0.86	0.85	0.87
AP	0.93	0.93	0.92	0.94

Note that these absolute numerical results do not estimate potential performance of these NIDSs in a live infrastructure, as we do not attempt to fully model the open-world detection environment. That would require at the least modeling statistical properties of modern networks in terms of attack's base rates and also appropriate training, validation and test data. Furthermore, an open-world NIDS deployed in a real environment would hardly base its prediction exclusively on such algorithms, but rather involve them in a more sophisticated pipeline tailored to the expected threats and other desired requirements. Arguably, the numbers in this study (and other studies that utilize benchmark datasets) can only be seen as relative performance indicators that allow us to compare different algorithms in isolation.

Nevertheless, the experiments show that deep learning-based algorithms reach comparable performance with a dense autoencoder on statistical features (one of the state-of-the-art approaches in applied ML-based security [133]). While other traditional ML algorithms could be used as baselines, and the experiments could show different results on different benchmark datasets, the trend we observe is promising in terms of the potential of feature learning for network intrusion detection, and deserved further investigation.

Our experiments with supervised NIDSs confirmed the trend: all four DNNs - TCN, 1D-CNN, LSTM and BiLSTM - reach comparable performance to the baseline. Random Forest that leverages manual features outperforms deep learning on this dataset: its demonstrated F1-score is consistently higher, 99.8% vs 99.1% (a top result by an BiLSTM).

Supervised Closed-World Detection. The state-of-the-art LUCID model (which is based on a 1D-CNN) has been previously evaluated only in a closed-world binary classification setup, distinguishing benign and DDoS traffic. We apply LUCID preprocessing on the whole dataset to obtain time-window based traffic representation for all attack classes. The multi-class model on time-

window based sequences (including UDP and ICMP traffic) obtains over 99% F1-score.

For the rest of the study, we will focus on the closed-world detection (classification) as the main task for a NIDS model. We choose 1D-CNN and TCN classifiers for the explainability investigation, based on their high training efficiency and the relative ease of interpreting the convolution mechanism. We consider three types of traffic representations as inputs: our proposed connection-based, LUCID time window-based, and nPrint binary time window-based.

6.3.4 Limitations and Open Questions

While deep neural networks on raw representations of the given dataset obtain better or comparable performance to Random Forest in these basic benchmarks, there are multiple open questions concerning the advantages of deep learning in this use case. The obtained numerical results do not confirm by default that the features automatically derived by the model are high-level temporal features. In other words, at this point, it is unclear whether the model benefits from automatic feature extraction (representation learning) or achieves high performance by using the input features directly, supported by the high predictive powers of neural networks. Therefore, while the pipeline is tailored to extracting temporal patterns from data and learning salient features automatically, the results are ambiguous as to whether sequential feature learning takes place and contributes to successful performance.

Since there is no direct way of interpreting the learned latent traffic representations consisting of abstract traffic features, we experiment with altering the ordering of packets and observing the impact on performance. When randomly permuting the packet order in the test set, the models (TCN and BiLSTM) lose up to 30% in F1-score. However, if we randomly re-order packets in both training and test data, we can successfully fit both models to this new data and again achieve over 99% classification performance. We can make the following conclusions. First, the DNNs appear to understand temporal structures. If an important packet moves to an unexpected timestamp in the sequence, the models trained on properly ordered flows cannot make accurate predictions. Second, it is implausible that the models understand the protocols behind the flows, as we can successfully fit the model to a randomly ordered flow. And third, this experiment is inconclusive regarding the exact features automatically extracted from traffic traces: i.e., whether these features are temporal and complex or mostly simpler combinations of monitored header attributes.

This experiment illustrates a "failure scenario" of a deep learning-based NIDS: the model fits perfectly well to traffic with a meaningless ordering of packets. While it is not the learning algorithm that is at fault (deep neural networks are known to fit completely random data), this essentially questions the validity and utility of the derived features. In the following section, we discuss other, more practical failure scenarios. Together, these scenarios motivate the need for a critical investigation of explainability and error analysis of DNN models that operate on raw representations.

6.4 Failures of Neural Networks on Raw Traffic Representations

The previous section motivates a more critical exploration of the deep learning pipeline when applied to network intrusion detection. In this section, we analyze a series of practical scenarios that expose problems of deep learning-based NIDSs. The underlying causes of the problems may be many and may even be external to the deep learning approach itself – such as artifacts in traffic introduced in data collection. Nevertheless, we hypothesize that even such external problems can be exacerbated by the basic approach that is normally used to evaluate deep neural networks.

In what follows, we first demonstrate several failure scenarios of deep learning-based NIDSs that utilize the algorithms discussed previously in the chapter. Then, for each scenario we theorize about the possible causes behind the failures. Finally, before venturing into a practical failure analysis further in the study, we review existing approaches in the literature that could be adapted to failure analysis of deep learning-based NIDSs.

6.4.1 Failure Scenarios

Scenario A: Fragmentation. For any given TCP connection, the maximum size of a packet that can be sent is determined by the Maximum Transmission Unit (MTU). We set up an experiment to determine the effect of varying MTU on the performance of a LUCID classifier. In this experiment, we first trained the classifier on data that was collected in a network environment with a certain MTU, and then evaluated its performance on data that was collected in a network environment with a different MTU. The results showed that changing the MTU caused the performance of the NIDS to drop drastically, rendering LUCID almost completely ineffective. Since a change in MTU usually leads to a change in packet size – and thus changes the number of packets per time window

– it results in a distribution shift across the entire dataset, which would explain why the performance of a LUCID degrades so heavily under these changing circumstances. Establishing the exactly same MTU in our testbed solves the problem, and LUCID is able to recognize a generated DDoS attack with high accuracy.

This experiment demonstrates that the invariant patterns of attacks (which are not eliminated by increasing the MTU or changing other unrelated network properties) are not picked up by the model. Instead, LUCID memorizes very specific patterns for the network environment it was trained in (the CIC-IDS-2017 setup). This behavior is expected given the low variability of network conditions in the benchmark dataset. We call this a "failure" scenario, as it demonstrates a critical caveat of IID training and testing on security data, and further indicates that a deep learning model might obtain very high performance without getting any understanding of high-level attack patterns.

Fine-tuning the model on the instances collected in the new environment (the principle of transfer learning) improves performance, and training on both datasets combined from start allows to obtain top success rate. It is, indeed, reasonable to expect that the features a deep neural network learns from such two datasets will be more related to the underlying task.

Scenario B: SYN Flood Experiment. When manually analyzing a separate NIDS dataset collected for DoS detection (CIC-DDoS2019[3]), we observe that the tool used to simulate the SYN flood attack in the dataset uses a fixed payload size. This feature alone could be sufficient to distinguish the SYN flood attack traffic from benign traffic, while the size of payloads is semantically not a discriminative feature for this type of malicious behavior. Indeed, SYN flood attacks are a type of a Denial of Service (DoS) attacks which work by repeatedly sending initial connection request (SYN) packets to the victim without completing the handshake. This would overwhelm all available ports on the victim and effectively consume all resources at the victim server. Therefore, the expected discriminative behavior to be recognized by a model within the context of one flow is a single SYN packet, regardless of its length.

To test this potential shortcut, we create a setup with a simple HTTP victim server and simulate the SYN flood attack with the *mausezahn* tool.[4] Benign traffic in the CIC-DDoS2019 dataset is not too heterogeneous, as it mostly includes SSH, FTP and HTTP protocols. Thus we can generate benign traffic by using automatic page requests, which makes benign traffic less diverse but should still fit what the model has learned from the dataset. For malicious

[3] `https://www.unb.ca/cic/datasets/ddos-2019.html`
[4] `https://man7.org/linux/man-pages/man8/mausezahn.8.html`

traffic, we generate SYN flood flows both with the same fixed payload size as in the dataset and with a random payload size.

In contrast to our intuition, LUCID trained on CIC-DDoS2019 obtains 100% detection accuracy on both datasets, varied-length and fixed-length. We apply filter activation analysis to the underlying 1D-CNN and compute general feature importances for two classes, which are listed in Table 6.4.

Table 6.4: Result of filter activation analysis on LUCID for a dataset with benign data and a random-length SYN flood attack, i.e., with varied payload sizes.

	Feature importance	
Packet feature	**Benign**	**DDoS**
Relative timestamp	0.00000	0.00000
PktLen	0.00011	0.00087
HighLayer	0.00201	0.00204
IPFlags	0.06855	0.00000
Proto	0.02001	0.01799
TCPLen	0.00000	0.00079
TCPack	0.00000	0.00000
TCPflags	0.00001	0.00001
TCPwin	0.11572	0.01920
UPDlen	0.00002	0.00000
ICMPtype	0.00000	0.00000

We can clearly see that the two key attributes used by the network to separate the two classes are IP flags and TCP Win (TCP Window Size). Although we expected Pkt Len (or TCP Len) to be the main attribute for the classification, the network has found another way to achieve the goal. After inspecting the traffic traces, we found that all the attack packets have a TCP window size of value 10000. Furthermore, all the attack packets have the IP flags value set to `0x0000`, while the majority of benign packets have the Don't Fragment bit set, resulting in `0x4000`. However, 1 percent of benign packets have a IP flags value of 0x0000, which is probably the reason why this attribute is less important than TCP Win.

This experiment serves as a perfect illustration of data collection artifacts creating trivial faulty correlation or causality links. The CNN finds the shortest path to connect the label with the observed patterns, which is in this case the combination of just two header attributes. The experiment also illustrates that human intuition based on domain knowledge and manual analysis can be a limited asset, even in a simple setup like binary classification. Automated

explainability and error analysis framework appear indispensable in analyzing models and data, especially in the presence of unexpected artifacts.

We believe that the susceptibility of neural networks to artifacts in data and the common practice of IID-only testing obstruct NIDS research in terms of obtaining reliable and generalizable outcomes. This problem needs to be addressed to establish new practices of reliable evaluation. In what follows, we review some key proposals in the literature related to the problem, in order to outline our vision for the path forward.

6.5 Related Work

Here we review the literature related to understanding and potentially resolving failures of deep neural networks in cybersecurity applications.

The ML community is currently coordinating significant efforts towards developing eXplainable Artificial Intelligence (XAI) techniques, including for interpreting deep learning models.[5] As there is no mathematical definition of explainability, this property is an elusive target that heavily depends on the application domain and human comprehension of concepts. Explainability can be assessed as follows: the higher the explainability of a model, the easier it is for a human analyst to understand why certain data triggers certain predictions.

The community recognizes several types of explainability [61]: explainability-by-design, post hoc model-specific explainability, and post hoc model-agnostic explainability. ML algorithms can be self-explanatory, such as decision trees, linear models, and rule-based models, or can only be explained with post hoc methods. Deep learning is in the second group: explanations of deep learning predictions can be derived by either examining internal model structures and parameters or applying independent analysis tools that only analyze network outputs and not their internal values. These two approaches are known as *white-box* and *black-box* explainability methods, respectively.

Warnecke et al. [230] have reviewed and evaluated white-box [16, 204, 190] and black-box [167, 121, 83] explainability methods for DNNs utilized in cybersecurity fields, such as malware detection and vulnerability discovery. Upon extensive experimentation, the authors concluded that white-box explainability approaches, also known as deep explainability methods, are consistently superior to their alternatives, owing to the access to a model's internal information. The authors show how white-box explanations allow them to detect several cases where features unrelated to security strongly correlate with the ground truth,

[5] `https://www.darpa.mil/program/explainable-artificial-intelligence`

enabling faulty causality. Similar to our use case, these emerge from artifacts in the underlying data. We draw inspiration from this analysis in selecting appropriate algorithms for explaining NIDS predictions. Specifically, we find that Integrated Gradients (IG) [204] provide the most accurate and granular explanations of predictions on our data.

A recent study by Arp et al. [13] on advisable and non-advisable practices in ML-enabled security has also demonstrated spurious correlations undermining the same state-of-the-art ML solutions. Specifically for vulnerability discovery, they detail their approach to finding shortcuts. They deploy the Layerwise Relevance Propagation (LRP) method [16] to compute relevance values for each code token and pick the top 10 most frequently occurring ones. This high-level analysis proves sufficient to determine which tokens in training and test data the DNN overfits to. It is important to note that this frequency-based analysis will not easily reveal less prominent shortcuts (that affect only a small number of samples) and non-spuriously correlated but overemphasized features. Overemphasized features will appear relevant to the task to a human analyst. Yet, they may be detrimental enough to cause the model to fail when deployed in a real environment.

Both studies employ state-of-the-art white-box explanation frameworks to find shortcuts; however, the frameworks are not tailored to find erroneously picked-up correlations. We note that the search for shortcuts relies on manual post-processing and heuristics. The search principle depends on the type of data and the exact problem statement, as the factors that cause artifacts are highly domain-specific. Therefore, it does not seem possible for the community to devise a universal systematic approach to searching for artifacts and related shortcuts in security data. It appears that for each new application, the search methodology must be designed according to the problem and data. Thankfully, this process can be supported with established explainability algorithms.

Apart from the dedicated explainability frameworks, where each instance can be analyzed, feature importance is another approach to understanding ML models. Feature importance is straightforward to compute, as it is a general statistic: feature importance methods assign a relevance score to each input feature for the whole class. This method may be an informative complementary technique that can act as a sanity check of a trained model. However, it does not provide enough insight into individual traces. Furthermore, feature importance relies on computationally intensive brute-force perturbations, which can be infeasible on large datasets and sophisticated models.

We conclude from this overview that neither explainability nor feature importance methods are sufficient by themselves to perform a systematic and optimal search for shortcuts. We believe that an appropriate error analysis

methodology for each application can be developed using explainability and feature importance as tools. Our vision for such a solution for the NIDS landscape is a methodology (or framework) that integrates the models, varied data representations, visualization of samples and corresponding explanations, and custom automated analysis functionality on arbitrary subsets of data. This methodology would allow a systematic evaluation of domain-specific heuristics and a guided search for potential shortcuts within the IID evaluation context. Our study thus presents the first steps in this direction.

6.6 Methodology

The shortcuts exposure methodology we design and evaluate in this study aims to expose and understand the failures of DNNs on IID data. While extensive OOD testing is the most effective way to verify shortcuts [77], real-world OOD security data is not readily available, and purposefully generating it in a helpful manner requires a clear objective. Our main assumption is that the suspected shortcuts found in the IID testing stage can inform the collection or generation of OOD data that could check for the most likely failure behaviors of the model in a real environment. Specifically for this study, we draw the line between IID and OOD as follows. We assume that the designer of a NIDS is aware of all possible attack strategies (a closed-world assumption) and possesses extensive annotated data (supervised learning). However, even though all attack categories are present in the training data, the samples themselves are not representative or diverse enough, so a slight modification to an otherwise recognizable attack sample would fall out of distribution to the model. Therefore, our analysis does not target OOD in the form of pure novelty (e. g., concept drift in benign traffic, zero-day attacks, or completely non-intended inputs). Instead, we define OOD as mere deviations in data that still respect the semantic meaning of network behavior already present during training but differ from training samples in irrelevant features or a minimal subset of relevant features. The fact that even such seemingly trivial changes force the test data to fall out of distribution for a model trained on a state-of-the-art benchmark dataset exemplifies the severity of failures considered in this study.

In this analysis, domain knowledge is expected to be crucial for assessing model behavior. A feature can be considered spuriously correlated or overemphasized only based on its perceived semantic connection with the label and domain experts' understanding of the underlying problem. Because domain knowledge is by design not thoroughly encoded in raw data representations, we rely on visualizations to make explanations of model prediction human interpretable.

At the start of the section, we discuss the NIDS dataset we use as the main example for developing our approach. Then, we detail the two parts of the methodology: the visualization framework and the automated analysis component.

6.6.1 CSE-CIC-IDS2018 Dataset

The CSE-CIC-IDS-2018 dataset[6] was created by the authors who also created CIC-IDS-2017. The main difference is that this dataset was generated on a much larger testbed environment, and contains a few more attacks than its predecessor.

Correction of the Dataset. The recent study by Liu et al. [119] shows that the artifacts and labeling issues present in CIC-IDS-2017 are also present here. Additionally, while there is a large volume of attack traffic, they find that the variability within attack classes was extremely low: the attack tools were only ran with one set of parameters, and the defenses set up on the victims (e.g., the Apache Web Server) were also kept the same at all times. This makes it much easier for a classifier to distinguish features that happen to correlate with attacks launched with a specific set of parameters inside a specific network environment. In one of their experiments on high-level statistical representations, it was sufficient to only slightly change the arbitrary parameters of one of the attack tools featured in the dataset in order to completely bypass a NIDS classifier.

In this work we use the dataset with corrected labeling and flow segmentation.[7] However, the underlying problems of the data collection setup behind the dataset cannot be overturned through post-processing. Therefore, this dataset presents an opportunity for developing a methodology for exposing the remaining shortcuts.

Note that we have experimented with three datasets in total (including the DDoS focused CIC-DDoS2019), but we report the analysis on CSE-CIC-IDS2018 as it is more recent and diverse than CIC-IDS-2017, and because the fixed labeling logic for this dataset is known (as opposed to the CIC-DDoS2019 dataset with only DDoS traffic).

Description of Classes. For the purpose of a more contained discussion, and a more straightforward comparison with prior work (such as LUCID), we focus

[6] `https://www.unb.ca/cic/datasets/ids-2018.html`

[7] `https://intrusion-detection.distrinet-research.be/CNS2022/Dataset_Download.html`

our evaluation on DDoS and DoS attacks. We provide a short description of the relevant attack classes, including the conceptual strategy behind the attack.

DoS Slowloris. DoS Slowloris aims to overwhelm a victim, usually a web server, by sending partial HTTP GET requests without ever completing the request itself, with these partial requests being sent at increasing inter arrival times. The Slowloris tool that was used for this dataset accomplishes this by sending an initial incomplete GET request followed by subsequent fragmented requests that only contain the HTTP header "`X-a`" with a corresponding value randomly chosen between 1 and 5000.

DoS Hulk. DoS Hulk attempts to overwhelm a victim web server by sending multiple HTTP GET requests that each ask the victim to keep the corresponding TCP connection open for upwards of 100 seconds. Sending many such requests would technically exhaust the victim's connection pool and render it unreachable. However, due to an issue with an underlying Python library that CIC used for the Dos Hulk tool, the attacker never requests to keep the TCP connection alive, and thus the effect on the victim web server is minimal.

Dos GoldenEye. This attack leverages the same attack vector as DoS Hulk, but does not suffer from any implementation mistakes. However, due to the victim web server running a specific Apache Web Server configuration, the TCP connections opened by the DoS GoldenEye tool are terminated after an idle timeout of 5 seconds, rendering the attack (somewhat) less effective. Note that this 5 second timeout is a good example of an attack artifact that is caused by the specific setup of the testbed architecture.

DDoS LOIC. DDoS LOIC is a tool that can launch TCP, UDP or HTTP packets against a victim server in order to overwhelm it. As opposed to the DoS tools previously mentioned which exploit some kind of vulnerability in the HTTP protocol, DDoS LOIC relies on sheer volume of requests in order to succeed. This dataset features traffic from both the HTTP and UDP versions of this attack, launched by 10 different attacking hosts.

DDos HOIC. The successor of DDoS LOIC. DDoS HOIC is only able to send HTTP GET and POST requests. Here also the tool was launched by 10 attackers simultaneously.

In our preliminary evaluation, we discuss the results based on a multi-class LUCID, trained on these attack categories and benign data. Note that we used the same architecture that was tuned on the non-corrected version of the CIC-CSE-2018 dataset in prior work, as it already reaches over 99.9% F1-score on our data.

Raw Traffic Representation. This dataset was designed for reasoning on the level of abstraction of statistical flow features, and not on the level of raw packet header attributes and their sequences. To apply deep learning models, we first reconstruct the exact packet sequences and label them according to the corresponding flows. We do so for two types of representations: TCP connection-based and time window-based (LUCID).

Moving from statistical features to raw representations not only increases dimensionality and granularity of inputs, but also allows certain traffic attributes to resurface which were originally not meant to be directly used in learning. These could be spuriously correlated attributes, i.e., semantically not associated with the label. Such spurious correlations can to some extent be detected during extensive data exploration by analyzing correlations of input features with class labels. However, another type of shortcuts is a truly correlated but overemphasized attribute, which carries such homogeneous values that the model may choose to simply utilize this attribute and remain blind to other influential more complex features. Our conjecture is that this type of a shortcut is more challenging to detect and cannot be easily reasoned about without training the models on raw time-series.

While comprehensive controlled OOD testing is the ultimate approach to exposing both types of shortcuts, we recognize that it would be beneficial to be able to highlight suspected shortcuts at the initial stage of IID testing. Our assumption is that the results of this analysis would allow to guide the subsequent OOD testing experiments.

Next in the section, we describe the visualization analysis framework we have built for manual exploration and explanation of neural network predictions on IID data. While our experiments focus on CSE-CIC-IDS2018, this methodology would apply to any research dataset, as generating optimal representative traffic data (both benign and malicious) free of artifacts is an open problem in the domain.

6.6.2 Visualization Analysis Framework

Our goal is to provide a practical solution to analyzing deep neural networks trained on NIDS data. The methodology is meant to simplify the workflow of designing, explaining, and troubleshooting research prototypes. With that in mind, we propose the first step that allows performing sanity checks over the models and underlying benchmark data, i.e., without necessarily having access to suitable OOD data. We propose to utilize a Visualization Analysis (VA) framework with a GUI that not only builds upon the most relevant advances in the literature for white-box explainability of DNNs (using the algorithms

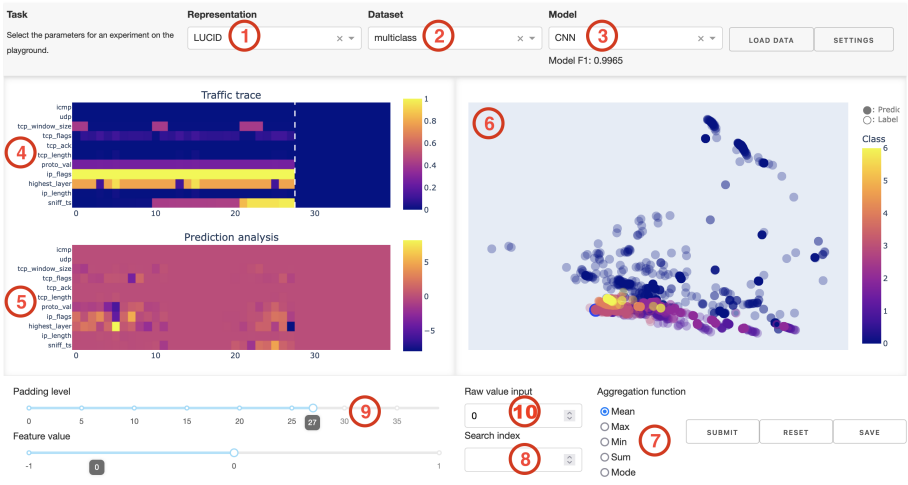


Figure 6.2: Functionality of the visualization framework for network intrusion detection neural network models.

provided by the Innvestigate tool [5]) but also is tailored to network traffic classification in the context of a NIDS.

The current purpose of the VA framework is to provide the following primary functionality:

- Visualization of 2D data embeddings, together with the results of classification for a given model, the confidence of prediction and the ground-truth labels.
- Visualization and manual analysis of white-box explanations per trace, with Innvestigate used as the backend.
- Manual search for or verification of suspected spurious correlations or overemphasized features, guided by the above two visualizations and the provided basic statistical analysis of arbitrarily selected data regions.
- Simulation of model failures through introducing specific (manual) perturbations in data, based on the above analysis.

Using the VA framework and domain knowledge, one can reveal useful heuristics for automating the search for shortcuts and other potential failure modes of a NIDS deep learning model. These heuristics can also inform the stage of real traffic collection for OOD testing down the road to contrast the shortcuts in the IID data.

Our visualization framework supports arbitrary models as long as they are shallow or deep neural networks to which the Innvestigate tool is applicable. The framework also allows switching between various traffic representations, including decimal or binary features, and with different approaches to segmentation. The length of traces for analysis and inference can also be dynamically varied inside the GUI.

While the framework provides a large space of possible combinations of influential factors, in this preliminary report, we focus on the LUCID model and its choice of traffic representation. Namely, we utilize a 1D CNN and learn from traffic flows segmented based on a predefined time window (10 seconds) and represented with 11 numerical header attributes on each timestamp. The benefits of using this representation are as follows: first, low input dimensionality makes visual interpretations of heatmaps more straightforward. And second, segmenting traffic flows based on time and not based on network connections is a promising ground for shortcuts, given the over 99% F1-score. We use this experimental setup to demonstrate shortcuts exposure further in the section.

Regarding the operation and structure of the tool, Figure 6.2 illustrates the major components:

1. Traffic representation format to be used.
2. Choice of the classes to be analyzed. The selection of a particular class in conjunction with the chosen representation will determine the dataset to be used.
3. The model that will be used to generate the predictions and the gradient-attribution analysis. The model selected is dependent on the representation and class determined previously.
4. Visual representation of the selected traffic trace, consisting of a matrix of features and timestamps of the trace. The specific values of each tuple (feature, timestamp) can be observed by hovering over them, as well as the raw value corresponding to the non-normalized traces. The visualization is dependent on the type of representation selected in (1). Each tuple (feature,ts) can be modified by clicking on it, using the input described in (10).
5. Visual explanation of the prediction for the selected trace, specifically a representation of IG [204] (however, other explainability algorithms in Innvestigate are available). Positive values indicate relevance towards the given prediction, while negative values indicate relevance against the prediction. The explanation is recalculated each time the source trace is modified.

6. Bi-dimensional embedding of the chosen dataset. The embedding is a visual representation of the model output in the penultimate layer (before prediction), processed by means of a dimensionality reduction through Principal Component Analysis (PCA). Each datapoint in the embedding corresponds therefore to a trace, where the color of the point determines the class to which it belongs and the color of the border the ground-truth class. Display of the trace and the gradient attribution method in (4) and (5) can be obtained by clicking on its representation in the embedding, and multiple traces can be selected to compute an aggregate trace. Moreover, prediction confidence, predicted and ground-truth label, and identifying index of the trace can be obtained by hovering over it.

7. The function to be applied to obtain an aggregate trace when multiple datapoints are selected in the embedding. This trace will be computed by element-wise application of the given function.

8. Search input using the trace index. Each index corresponds to the index of the trace in the selected dataset and is currently used as a unique identifier.

9. Slider to change the amount of padding in a trace by determining from which timestamp 0-padding is applied. In case of removing padding, the last non-zero packet of the trace will be repeatedly replicated.

10. Numerical input to modify the trace by assigning a new value to a selected tuple (feature, timestamp), as described in (4) .

6.6.3 Automated Exposure of Shortcuts

This part of the methodology builds upon the manual exploration facilitated by the VA framework. Its purpose is to verify the suspected shortcuts by means of actively perturbing the values of suspicious attributes and packets. The assumption here is that a shortcut value, while being not semantically connected with the class, makes the model so fragile that a non-optimized perturbation (as opposed to adversarial attacks) will be sufficient to change the label with a high success rate.

For this step, we develop a script that provides the following functionality:

Feature Ranking. Computing Innvestigate outputs (heatmaps with positive and relevance values) over the whole dataset. Clustering the relevance values with the K-means algorithm to determine the likely overemphasized features.

Value Frequency. Computing values distributions per attribute and/or per packet or both in a chosen class or across the dataset.

Automated Perturbation. Perturbing the specific attributes in specified timestamps with a provided list of values in a chosen class or across the dataset.

Next, we discuss the preliminary results of the analysis.

6.7 Preliminary Evaluation

In this section, we report on some of the findings that emerged from applying the general methodology described above to the corrected CIC-CSE-2018 dataset. The discussion is focused on one combination of a model and raw traffic representation: a 1D CNN applied to time window-based LUCID features (time window equals 10 seconds, maximum amount of packets in a flow segment – 40, padded with zeros).While a thorough systematic exploration on all considered models and representations is pending, we present preliminary evaluation results of the framework. The focus is on shortcuts exposure in the IID testing stage. The experimental results presented here illustrate the utility of the VA framework and how automated perturbation analysis based on relevance values that Innvestigate computes can guide the search.

To illustrate the methodology, in what follows, we briefly outline the patterns we found in one of the attack classes – DDoS HOIC.

Observation 1: IP Flags. Our explainability analysis on a subset of the dataset shows IP flags across several timestamps as one of the most relevant features for the vast majority of the class samples. However, this observation contrasts with the distribution of values of the feature, as most of the traces exhibit the same configuration of IP flags: corresponding to only the DF (Don't fragment) flag active across all timestamps. The only exception to this value distribution is given by a small subset of benign traces where the DF flag is disabled intermittently. Given that there are only two different values for the flag in the studied examples, one possibility is that the model uses this feature as a binary signal to detect spatial characteristics of the trace, such as trace length, or particular patterns of intermittency, regardless of the specific value. Our preliminary evaluation seems to point in this direction, as disabling the DF flag in the last packet of DDoS HOIC samples causes 99.35% of the class to be predicted as benign. Moreover, regarding the benign-specific intermittent

patterns, we observed in our experiments that disabling the DF flag to match these patterns in other classes is often enough to flip the predicted label of the samples to benign.

Observation 2: TCP Window Size. When considering DDoS HOIC samples, the TCP window size of the first packet is consistently ranked as the most relevant feature for the predicted class. Moreover, it appears that a model can identify DDoS HOIC exclusively through the TCP window size of the first packet. The specific value, 65535, is found in 99.73% of the DDoS HOIC samples, and its prevalence is negligible in the rest of the dataset, as only 0.58% of benign samples also exhibit this value in the first packet. As such, substituting this value alone turns out to be powerful enough to flip 99% of the class to DoS Hulk. We find that the exact value we inject in place of the original TCP window size does not matter for the classifier: as long as it is not 65535 in the first packet, the swap is sufficient.

Observation 3: Protocol Values. Most of the samples considered in the experiments correspond to HTTP traffic, except for a small percentage of benign traces that exclusively consist of DNS traffic. In such instances, our analysis reveals the feature concerning the protocols as the most significant one in isolation, given that the model trivially identifies the presence of the DNS/IP/UDP stack as a key indication that a sample is benign, as opposed to the rest of the HTTP traffic. Although impossible in practice, since in the constructed sequences, all packets share the same transport protocol, injecting a DNS packet into any DDoS HOIC trace at specific timestamps will result in the sample being predicted as benign. A similar outcome can be achieved in practice, however, given how protocols are encoded. In the chosen data representation, the protocol stack is given by the decimal value of a binary string listing the presence of a set of protocols in an arbitrary order. As the model only notes a numerical value, introducing a packet in which the protocols map to a decimal representation similar to that of the DNS/IP/UDP stack is sufficient to flip the original label to benign.

Observation 4: TCP Flags and TCP/IP Lengths. In addition to our guided exploration using gradient-based feature relevance methods, we launch an automated analysis that perturbs attributes regardless of their perceived importance. The perturbation substitutes each particular value of a feature and a timestamp with the most frequent values found in other classes. Using this heuristic, we detect that specific changes in the combination of three features, TCP flags and IP and TCP lengths, cause impactful changes in prediction, whereas altering each feature in isolation is not enough to flip the label. Upon closer inspection, we observe that

most of the analyzed samples have the congestion-related TCP flags CWR and ECN enabled, a behavior only present in the DDoS HOIC class, in a similar fashion to the previous scenarios. Semantically, congestion flags are a characteristic of the specific network environment conditions and are not related to the attack. As such, removing the CWR and ECN flags in the first packet, in conjunction with modifying the seemingly non-relevant values of IP and TCP length in a specific timestamp, is enough to flip nearly 100% of DDoS HOIC samples to Benign. This unintended effect is a suspected shortcut induced by a data collection artifact, which relies on a combination of three specific input packet attributes on specific timestamps.

To summarize these observations, we methodologically detect several distinct suspected shortcuts for the DDoS HOIC attack, where both types of shortcuts are involved – spurious correlation (congestion flags, TCP window size...) and overemphasized features (specific protocols signaling benign behavior). Moreover, we prove that the existence of shortcuts may not correspond exclusively to specific values of individual features. Data processing, or even the choice of the representation format, may also lead to a different set of shortcuts, sometimes even coexisting simultaneously with artifacts of data collection.

Analyzing cases of model overfitting to a single highly relevant feature seems to be an effective screening strategy for detecting shortcuts. Further examination of the existence of the shortcut can be attempted by perturbing the value of the feature under consideration, effectively *removing* the shortcut in order to observe its effect on the output of the model, or by evaluating the effect of utilizing that value in other traces, effectively *introducing* the shortcut in an effort to determine a change in the prediction. While the first approach appears to be more effective, the strategy for selecting the exact perturbation values remains an open question. For now, it appears that highly relevant common values of other classes often work best.

Furthermore, one of the suspected shortcuts seems to depend on a combination of 3 features. This case illustrates that some shortcut learning instances might only occur when several overemphasized or spuriously correlated features are simultaneously present. This observation puts forward an assumption that gradient-based computations of feature relevance are not entirely sufficient to detect all spurious correlations in data. The brute-force perturbations we deployed do not provide enough insight into how these complex multi-feature correlations could be detected beforehand in the absence of high-relevance values. As such, this problem demands further research.

Overall, we confirm that visualizations of traces and their explanations, together with the embeddings, can facilitate a targeted search for suspected shortcuts.

Using the insights collected in this stage can thereafter either inform the OOD testing, or be sufficient in its own to improve the data, representations, and even models.

In the next section, we discuss how we could extend this methodology in future work in terms of shortcuts exposure, and other future research directions.

6.8 Discussion and Future Work

6.8.1 Exposure of Shortcuts

In future work, we will extensively evaluate the analysis framework over different DNNs, attack classes, and available raw traffic representations. This systematic exploration will allow us to isolate the impact of each component and design choice of a deep learning-based NIDS concerning its vulnerability to shortcuts in the dataset.

As the next step, we will expand the methodology beyond exposing shortcuts in the IID stage to (automated) OOD testing - a universal solution to shortcut learning [77]. Naturally, obtaining OOD data can be very challenging in the security domain: often, it only presents itself in the live deployment of the system. Moreover, annotating such real-world data can be a challenge in itself. In lab conditions, OOD data can be obtained on two levels: (i) generating numerical OOD samples that trigger the suspected shortcuts and (ii) testing on realistic (or live) traffic outside of the distribution of the data used for the NIDS design. One such example scenario was shown in Section 6.4, where the SYN Flood attack was regenerated in a live network with varied packet lengths, which created OOD data and undermined the detector, despite the attack remaining the same. We designed this experiment based on manual analysis of attack traffic and understanding the flaws of the tool used for the attack's generation. However, without similar intuition, exhaustive perturbations of each attribute can be infeasible or even meaningless. The methodology needs to provide a more systematic approach to obtaining or generating such OOD data, for example, by building up on the heuristics and the suspected shortcuts detected in the IID stage.

6.8.2 Prevention of Shortcuts

Ultimately, this research direction aims to design systems that avoid learning shortcuts and instead learn the intended underlying patterns. Since the definition

of a shortcut depends on the human semantic understanding of the problem and the context, this is not a trivial task and is shared by all applied AI domains. One general path towards the goal is collecting more representative data of good quality that does not suffer from collection artifacts. Next, considering several datasets at once is a recommended practice: it often allows to shed light on oversights in individual datasets and the corresponding models. Also, the choice of data representation seems extremely influential: when choosing a particular level of abstraction, one may involuntarily introduce a shortcut that the downstream model would pick up. And finally, it appears sensible to assume that different learning models or algorithms and their parametrizations will exhibit different degrees of vulnerability to shortcuts. After all, sensitivity to shortcuts can be directly related to underlying learning mechanisms and the generalization abilities of models.

We list several possible investigation actions that could diminish the vulnerability of deep neural networks to shortcuts in data, building up on the analysis by Geirhos et al. [77]. While these approaches apply to any kind of neural network, it is with deep neural networks that explainability analysis is especially challenging. In terms of the choice of specific network architecture, prior research hypothesizes that using convolutions makes it harder for a model to use the exact locations of the features within the input sample as the main attribute for solving the task [49]. This study shows that convolutional networks require further help to avoid shortcuts, but there is no evidence of how they compare to other deep learning architectures. Possible solutions could be inspired by research on making deep neural networks robust against adversarial perturbations. For instance, similarly to adversarial training, training on more data could be successful at preventing the model from learning shortcut approximations, given that the new data refutes the shortcuts and/or valid patterns that are easier to learn. Moreover, it appears possible to modify training data so that specific shortcuts are blocked and, simultaneously, the vulnerability to adversarial perturbations is reduced [124]. From the perspective of learning parameters, the commonly used loss function – cross-entropy – can lead the DNN to learn a simple solution by taking shortcuts. Changes to the loss function could force the DNN to use more available information by penalizing overemphasized features [99]. Conditional variance regularization [88], which uses additional information about the training data, can also force the model to use the intended features, improving predictive accuracy even in the presence of shortcuts. Similarly, invariant risk minimization [12] can help capture invariant correlations to enable OOD generalization, assuming access to feature values from different training environments. Moreover, it has been shown that the learning rate influences which patterns networks focus on: depending on how large or small the learning rate is, it can either pick up simple patterns or perform more complex pattern learning and memorization [14, 116]. Overparametrization – which deep neural

networks are infamous for – has also been linked to higher vulnerability to spurious correlations in data, despite the increase in performance, especially in the context of data imbalance [176]. And in general, as was stated previously in the chapter, neural networks themselves are designed in a way that exercises simplicity bias. Research has established that optimization algorithms used in neural networks, such as the inductive bias in stochastic gradient descent and its variants, push the model towards learning simple functions, including spurious correlations [231, 55, 210, 201].

Given what seems to be an inherent susceptibility to shortcuts and the difficulty of reasoning about the semantic meaning of the internal representation of neural networks, deep learning from raw traffic representations remains a non-obvious choice for a NIDS. Arguably, these challenges supersede the question of robustifying such a NIDS to adversarial perturbations, as they undermine the overall validity and utility of the proposed model for intrusion detection – i.e., even without taking into account adaptive attacks against AI.

Therefore, future research has to address several fundamental aspects to establish automated feature extraction from network traffic with deep neural networks as a valid (and beneficial) approach to intrusion detection. In summary, this chapter touches upon the following such aspects, which are largely connected:

Benefits of Automated Feature Extraction. Analyzing traffic at the abstraction of sequences of raw packet headers allows the employment of feature learning but, at the same time, presents significant problems. Future research needs to further explore the need for automated feature learning from raw traffic in terms of its conceptual benefits, i.e., beyond convenience and efficiency. For example, this step can be done by demonstrating better adaptability to distributional shifts in network traffic compared to high-level flow statistics. However, this property can only be appreciated if a useful semantic meaning behind the extracted discriminative features can be retrieved. The retrieval of the semantic meaning behind novel patterns cannot be fulfilled without the next aspect in the list.

Explainability. As is already reasoned in the prior study on explainable deep learning in security [230], explainability functionality has to be integrated into a deep learning-based NIDS pipeline. This functionality would include explanations of made predictions and, ideally, an interpretation of how the model reasons about the problem as a whole. It would be extremely valuable to show human-understandable explanations for various classes of intrusions with an explicit connection to domain experts' intuition about the attacks.

Error Analysis. One of the purposes of the explainability functionality is the error analysis of a NIDS model. For trustworthy defensive systems, understanding and troubleshooting errors in predictions is paramount. Our study contributes to advancing error analysis in NIDS by focusing on the methodological exposure of shortcuts in the context of IID testing. But much more research is needed to systematically overview and analyze all failure modes of deep neural networks when applied to intrusion detection. One way forward would be to adapt known error analysis frameworks on images [192, 143] to time-series traffic data with discrete values.

NIDS Data Quality. All of the aspects listed above strongly depend on the quality of the used dataset. Continuing improving practices behind network traffic collection, attack and network behavior simulation, and labeling will contribute significantly to the domain across the board, including the research on data-driven solutions. Furthermore, there is a strong need for collecting research data specific to a particular open question, such as distributional shifts in network environments and adaptive attacks through adversarial perturbations.

Selection of a Robust Input Data Representation. Even with the best methods for data collection, artifacts might never be fully eliminated. Moreover, certain artifacts might be involuntarily introduced with a traffic abstraction chosen for the data input format. And finally, even if a certain feature is seen as spuriously correlated with one of the classes, completely removing it is hardly a solution: it may be correlated with other labels. Overall, optimizing raw representations of network traffic for deep neural networks is a valuable research direction.

Hardening Neural Networks. As per the above discussion in the section, deep neural networks show an inherent simplicity bias and, thus, higher vulnerability to shortcut learning as well as adversarial attacks. Exploring how deep learning models can be designed, parametrized, and trained to prevent learning from shortcuts in imperfect data would be a fundamental contribution to deep learning in security and privacy, where even rare incidents can have drastic consequences.

6.9 Conclusion

This chapter presented our work in the scope of flow-based network intrusion detection with a focus on applying representation learning from raw traffic representations instead of hand-engineered high-level statistical features. The primary outcome of this study is a critical analysis of limitations and open

questions about using deep neural networks as a defensive tool in network security, specifically for intrusion detection. Despite the very promising performance, comparable with or better than other data-driven baselines, the exact advantages of representation learning are yet to be conclusively demonstrated. At the same time, the potential pitfalls of using deep learning for this task are evident. In our study, we proposed a methodology to mitigate these pitfalls, i. e., to expose and explain shortcuts that the model learns from NIDS data, effectively revealing how they can be corrected. Our methodology includes a visualization framework and an automated perturbation library that we intend to make public to facilitate further analysis. Our contributions provide a stepping stone for understanding and mitigating the limitations of this powerful modeling technology in the context of network intrusion detection.

Chapter 7

Conclusion

> "A ship in harbor is safe, but that is not what ships are built for."
>
> — *John A. Shedd (Salt from my Attic, 1928)*

Today we are witnessing the systematic proliferation of deep learning into primary applied Artificial Intelligence (AI) domains, including the safety-critical cybersecurity and privacy areas. An integration of such a scale and nature demands empirical, methodological investigation in order to move towards a profound understanding and control of these powerful techniques. Furthermore, advanced data analytics through deep learning may yield novel insights into the application domain.

In this regard, the present dissertation explored the application of deep learning to improve security and privacy in two distinct application domains: traffic deanonymization and network intrusion detection. This chapter concludes the thesis by summarizing our results from a broader perspective, then reviewing the open challenges and overarching insights that go beyond our specific findings.

7.1 Summary of Contributions

The practical analyses presented in this dissertation contribute to fostering the sharing of experiences, insights, and best practices between various Machine Learning (ML) applications in cybersecurity. Specifically, our research on

applied deep learning has addressed both offensive and defensive aspects of cybersecurity and privacy. We analyzed deep neural networks' applicability, advantages, and disadvantages, first as attacks against privacy in Tor and second as defenses against network intrusions. In doing so, we contributed to assessing the severity of privacy threats in Tor, on the one hand, and the potential of end-to-end data-driven network defenses on the other. At the same time, this dissertation revealed deep neural networks' many complexities and possible limits, which resurface when attempting representation learning from raw security and privacy data.

In what follows, we summarize the main findings achieved for each application domain and recapitulate the main challenges that need to be overcome going forward.

Traffic Deanonymization. Our research on traffic deanonymization in Tor contributes to advancing empirical estimations of privacy leakage in the wild. First, in Chapter 3, we opened the door to more advanced data-driven website fingerprinting attacks, considering a deep learning savvy attacker. Our automated feature learning techniques have permanently refined the Tor threat landscape by relaxing the demand for manual traffic analysis knowledge, thus triggering an ample line of follow-up research on automated website fingerprinting attacks and defenses, active to this day [171, 67, 130, 215].

In the broader perspective, the results in Chapter 3 demonstrated that deep learning is not only more potent at side-channel traffic analysis but also shows higher resilience to concept drift in data, making it an even more compelling tool for an attacker. Interestingly, other studies have since made a similar observation, specifically in the context of malware detection. Andresini et al. [10] analyzed deep learning and traditional ML approaches in application to Android malware classification and showed higher resilience of deep learning to time decay in data in their evaluation setting. They also highlight the trade-off between resilience to concept drift and the lower explainability and higher training time. While this trend is not guaranteed to hold across all related applications, this phenomenon certainly demands research attention.

Chapter 4 introduced our second contribution to advancing the scientific assessment of privacy leakage in Tor. This work presented a more realistic multi-proxy data collection setup for end-to-end correlation attacks, with the purpose of bringing lab evaluations closer to reality without compromising the experience of Tor users. This critical study highlights the impact of data collection artifacts for deep learning-based attacks and contributes a general scientific methodology for tuning, replicability, and fair comparison of data-driven attacks.

Based on the main contributions in these two chapters on traffic deanonymization, we can now discuss priorities for future advances. In addition to the more fundamental research questions in this domain, where the simulation of representative activities in Tor, the accurate estimation of priors, and the lowering of the practical overhead of countermeasures are prominent concerns, in this discussion we focus on deep learning-related challenges. Firstly, future research directions should include the interpretability of deep learning-based traffic analysis in Tor: explaining and interpreting deep learning attacks should inform the composition of reliable and meaningful countermeasures. The first steps in that direction have been recently performed in website fingerprinting, where universal techniques for explaining convolutional deep neural networks have been leveraged [51]. Our work applies similar analysis to explaining end-to-end traffic correlation attacks, where we demonstrate the first results and present ideas for future research. Specifically, there would be high value in isolating timing, ordering, and packet length information, as well as upstream and downstream traffic, in terms of its actual correlation with the label, as opposed to the specifics of the traffic simulation environment, given that the deep model itself cannot know the difference. In order to inform countermeasures against deep learning for traffic analysis attacks, obtaining novel specific insights into the workings of deep neural networks on this data is required.

Secondly, and more fundamentally, we recognize the strong motivation behind considering protected network traffic as a novel data modality in AI, with its structure, unique properties, and corresponding context. This angle of view would prompt the community to rethink our approach to learning from this data: it is conceivable that the widely used convolutional neural networks, for instance, are not the ultimate answer to website fingerprinting or end-to-end traffic correlation, as they were initially designed with other AI tasks in mind. Future research cannot safely assume that real-world adversaries will stay bound by ML approaches designed for computer vision and speech or text processing. Furthermore, as our research shows, the standard training pipeline of deep learning does not entirely fit open-world traffic analysis tasks, at the very least due to their dual objective nature. Consequently, a perfectly fitted neural network on Independent Identical Distribution (IID) data is not necessarily the most efficient and robust attack in the open world. Therefore, completely novel architectures and learning algorithms may be proposed going forward, which would respect the properties of protected traffic and the specific requirements of the privacy domain.

Network Intrusion Detection. In the second part of the dissertation, we considered the field of intelligent network security solutions. We first contributed a comprehensive discussion on data-driven network intrusion detection systems

in Chapter 5, where we analyzed the interplay of ML techniques and demands for (open-world) network intrusion detection and identified pressing challenges and priorities for future work.

Next, in Chapter 6 we presented our final study on deep learning as a candidate algorithm at the core of Network Intrusion Detection Systems (NIDSs) with enticing properties awaiting systematic practical confirmation. Specifically, we focused on the applicability and reliability of deep neural networks as a stand-alone mechanism for analyzing raw traffic and detecting malicious or suspicious behaviors. To that end, we considered several designs, including those in the literature, and conducted broad experimentation on benchmark datasets. As a result of this exploration, we demonstrated that deep neural networks are inherently susceptible to misleading artifacts in data, which is a significant drawback when it comes to real-world defensive applications. We attribute this issue not just specifically to deep learning but to many components of the pipeline: data collection, the choice of data abstraction or representation, simplicity bias of neural networks, and the lack of Out Of Distribution (OOD) testing in standard evaluation practices, to name a few.

In the chapter, we outlined our vision for a future solution and experimented with what we see as a general methodology for mitigating this critical shortcoming in data-driven network intrusion detection. Thus, we wish to inform future research in modifying existing practices so that practical verification steps become not just mandatory but also standardized and easily accessible for NIDS researchers. For example, this could be realized through a collaborative, coordinated process for the joint development of tools, frameworks, and datasets tailored to this specific domain instead of relying on general-use ML libraries. We contributed one such potential framework, specifically for systematically analyzing the shortcut learning vulnerability of deep learning-based NIDSs. In our future research, we intend to construct a general failure-analysis methodology that focuses on exposing shortcut learning and addressing both the IID and OOD settings, which would empower the community to improve the validity of NIDSs' evaluations. In this respect, independently of and concurrently with our work, a highly relevant recent research study [98] has pursued a decision-tree-based analysis for a similar purpose, namely to detect instances of susceptibility to shortcuts. Future work can assess whether this approach based on simple interpretable models applies well to rich raw traffic representations and non-linear, highly parametrized models such as deep neural networks. An alternative path to search for shortcuts and blind spots in data-driven systems may be to build upon adversarial perturbations. It would be particularly valuable to analyze the interplay between adversarial robustness and analysis of intended model behavior in the presence of data artifacts. The trustworthiness of models thus combines two significant properties: how correctly it learns the intended

task and how well it withstands adaptive adversarial inputs. It is, hence, sensible to look at the joint solution space for both these properties in future research.

Furthermore, a promising future research direction would be to isolate the various components that impact the vulnerability to shortcuts and other failure modes, emphasizing the choice of data representation for network traffic. While there have been some studies [94] that proposed universal network traffic representations for a variety of tasks, our research indicates that the choice of traffic representation cannot be taken for granted if one wishes to avoid unspecified behavior while searching for patterns in data that are relevant to the given task. Therefore, considering the associated risks, an open question remains whether the universality of traffic representation in defensive security is achievable or even desirable. Perhaps, on the contrary, in order to ensure valid behavior of AI, it would be instrumental in stepping away from the optimistic end-to-end nature of data-driven solutions, even when involving representation learning from raw data through deep neural networks. This shift would entail returning the reins to domain experts in the matter of composing meaningful and robust representations of raw data for deep learning. Additionally, the dynamic nature of the network intrusion detection problem, and its neighboring tasks, means that a representation valid in a given context and environment may cease to be valid when the environment changes. This substantial risk also implies that domain experts are expected to monitor the system and reevaluate previous design choices. While deep neural networks are powerful at extracting novel patterns, they still rely on a predefined input representation to automatically extract an appropriate latent representation. Adapting the input representation may correct the system's behavior; however, this remains solely in the control of the system's designers.

7.2 Concluding Remarks

This dissertation reasons about and illustrates the value of researching different applications of deep learning in cybersecurity beyond the theoretical reasoning of its benefits. Specifically, our findings show that the same set of tools can reveal different strengths and weaknesses depending on the exact application and real-world deployment requirements. Namely, while we demonstrate deep neural networks' high suitability and superiority in offensive research, specifically regarding side-channel attacks, its utility as a universal end-to-end defensive mechanism remains open-ended.

Then the natural question is, what differs between these two sides of security and privacy research to such an extent that the same technology diverges

considerably in its behavior and effect? This observation may appear puzzling at first, considering that in both domains studied in this thesis, highly similar deep learning models are applied to highly similar data and perform essentially the same learning task, i.e., closed- and open-world classification of network traffic based on its metadata. Based on our work, we can provide the following general explanation behind this tension, which would likely apply to many other application domains in security and privacy.

Offensive vs. Defensive Applications. Offensive research seems to present a qualitatively different set of requirements: in contrast to defenses, attacks usually have a higher tolerance for errors and thus do not require strong performance guarantees to be successfully deployed in the wild. In other words, the definition of a successful offensive solution is more fluid. Contrary to that, intelligent data-driven defenses in safety-critical contexts are typically required to diminish the error rate and decrease the burden on operators; otherwise, the costs of involving ML become hard to justify. This aspect by itself already increases the expectations from probabilistic non-deterministic approaches such as deep learning when used for defensive purposes. Moreover, while offensive and defensive solutions both require explainability and interpretability properties to enable scientific progress, the urge to provide those in case of attacks is usually more minor. This difference is due to the inherent fact that even opaque attacks demand consideration by the community as they reveal vulnerabilities. In the meantime, a defensive mechanism has questionable utility for the community if it does not immediately provide insights into the detected malicious patterns. Furthermore, attacks may be targeted as well as untargeted, allowing for assuming specific knowledge about the deployment context. However, when designing defenses, this knowledge is generally either assumed to be unavailable or is assumed to be subject to changes. Examples of such deployment contexts explored in this dissertation have been specific network conditions, the size of the closed world or the number of behavior categories, and performance metrics of interest, to name a few. Finally, Kerckhoff's principle plays a significant role here: designers of defensive solutions ought to assume intelligent adversaries that are not only aware of the exact defensive mechanisms but may also continuously adapt to them. In contrast, a deep learning-based attack can be considered sufficient once it successfully undermines the victim system in its present condition, i. e. without an immediate evaluation of adaptive mechanisms. That is likely because specific defenses designed against ML or deep learning attacks (e. g., adversarial perturbations in traffic that aim to mislead the attack) may be infeasible, or at the least costly and impractical (such as when considered for the Tor network [130]).

In summary, we conclude that the suitability and success of deep learning

in offensive and safety-critical defensive applications may differ due to the disparity in terms of costs of errors, demand for explainability, targeted versus generalizable nature, and demand for adaptability. These observations and reflections may stem from the applications at the core of this thesis but are also relevant to other domains considered in this PhD trajectory, such as malware detection, authentication, host-based intrusion detection, side-channel attacks on authentication data, and more. Moreover, these aspects are not entirely bound to deep learning: while deep learning may exacerbate some of the limitations due to its complex nature and more demanding underlying learning tasks, the fundamental assumptions and limitations of ML are to be considered regardless of the chosen algorithm.

A Note on Generalization of Deep Learning Solutions. That being said, there is a clear trade-off between complexity and learning power, which informs the choice between deep learning and traditional ML approaches. Our work confirms that the superiority of deep learning can only be asserted for a specific application and a specifically chosen set of requirements or properties of interest. Depending on which guarantees the desired solution should provide, deep learning can be either a fitting choice or a non-intuitive solution that brings more challenges, possibly even including unintended consequences that could otherwise be avoided. However, existing evaluations of applied deep learning, such as those described in this thesis, can become essential in exploring and making correct design choices for future endeavors.

Research-Specific Aspects. Orthogonal to the remarks above, a special consideration for this line of work is research-specific aspects, such as quality of data, sound methodology evaluation, and reproducibility. We confirm through our investigation that in order to provide reliable findings, novel research on defensive deep learning should integrate and thoroughly evaluate the explainability of predictions, error analysis, and out-of-distribution testing components. This goal is crucial given the spread of imperfect datasets obtained through non-standardized and often opaque data collection. In this dissertation, we recognize the quality of security and privacy datasets and the reproducibility of deep learning solutions as two overarching practical challenges in enabling sound progress in data-driven cybersecurity. Across different applications, we contribute novel datasets or improve existing ones to enable progress in the field. Specifically, our research shows the high utility of new datasets in retrieving more reliable insights but also analyzes the remaining challenges in data collection and proposes directions for future improvements. Additionally, we tackle the inherent problem of reproducibility of deep learning research by open-sourcing the designed solutions and experiments, together with transparently outlining

the details and limitations of deployed evaluation methodologies. Understanding the underlying technology, as opposed to seeing it as a universal black box, is the foundation of approaching its evaluation more critically. To that end, cybersecurity and privacy fields can continue interacting with other applied AI domains and, more fundamentally, advances in theoretical AI research.

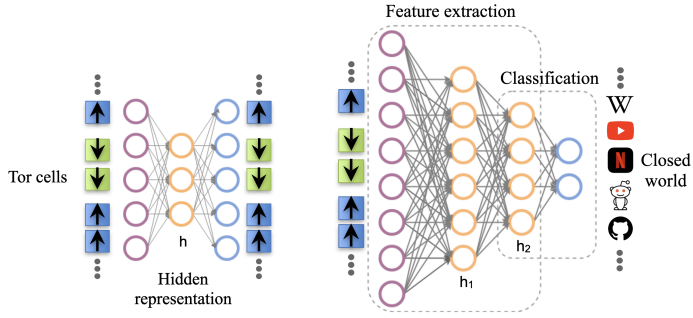
In conclusion, this thesis explores deep learning not as a silver bullet in cybersecurity and privacy but as a promising multifaceted technology worth supporting, advancing, and understanding. The provided findings and reflections contribute to the overall journey and offer a glimpse into the inherent size of the challenges ahead in this intriguing research landscape.

Appendix A

Additional Materials for Website Fingerprinting

A.1 Model Architectures

This appendix depicts the deep neural networks used in Chapter 3. The general learning algorithms are described in the background chapter on deep learning (Chapter 2). Figures A.1, A.2 and A.3. instantiate these architectures for the given task: classifying a Tor user's traffic traces to their website destinations.



(a) Autoencoder. **(b)** SDAE from two autoencoders.

Figure A.1: Stacked denoising autoencoder used for website fingerprinting. First the autoencoders are pre-trained for reconstruction of the exact traffic traces (a), then stacked together and fine-tuned for classification (b).

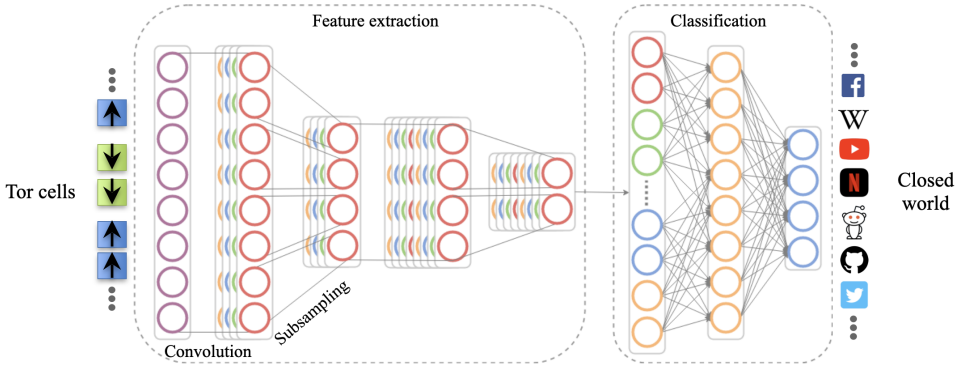


Figure A.2: Convolutional neural network for website fingerprinting. The feature extractor consists of two pairs of convolutional and pooling layers. The resulting feature maps need to be flattened and concluded by at least one regular fully-connected layer prior to classification. To avoid overfitting, dropout is applied and only two fully-connected hidden layers are used, which minimizes the number of learnable parameters.

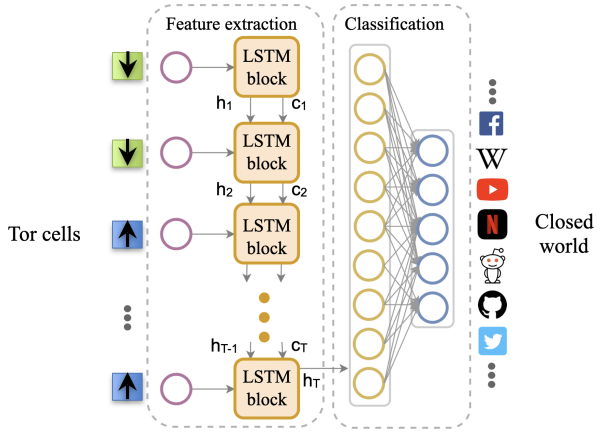


Figure A.3: Unrolled Long Short Term Memory (LSTM) network for website fingerprinting. Each Tor cell in an input sequence is processed sequentially, by one LSTM block at a time, and the block's output is used as an input to the next recurrently connected block. For website fingerprinting, we chain two hidden LSTM layers.

Appendix B

Additional Materials for Tor Traffic Correlation

This appendix lists additional evaluation materials for the end-to-end traffic correlation attack analysis presented in Chapter 4.

B.1 Original DeepCorr Performance

We list the search spaces and chosen hyperparameters for the original DeepCorr architecture in Table B.1. The hyperparameter tuning process of $\mathcal{DC}$ is depicted in Figure B.1, showing Average Precision (AP) of different configurations evolving over 30 epochs. Note that the runs are terminated early when they stop progressing. The longest runs take over 8 hours, so we set patience for early stopping to 5 epochs (vs. 10 epochs for $\mathcal{TDC}$). We refer the reader to our website in footnote 1 for more details on reproducing original DeepCorr on public and novel datasets.

B.2 Tuning and Training Details

Table B.2 lists 5 top tuning experiments for $\mathcal{TDC}$ models on both datasets (based on the search spaces defined earlier in Table 4.3). While the tuning framework sorts the runs by their final validation AP (i. e., at the last or early stopping epoch), we choose the best configuration based on the maximum validation

Table B.1: Hyperparameters of the $\mathcal{DC}$ model on the multi-proxy dataset and final chosen configuration for $\mathcal{TDC}$. Fixed kernel sizes for the first convolutional layer were (2, 30) and for the second (4, 10). Fixed strides were (2, 1) and (4, 1), respectively.

Hyperparameter	Search space	$\mathcal{TDC}_{SD}$	$\mathcal{TDC}_{MD}$
kernels	[2000,800]	[64, 128]	[64, 128]
pool size	{3, 5}	5	5
dense layers	{[2000],[2000, 500]}	[500, 100]	[500, 100]
optimizer	—	Adam	Adam
lr	{0.001, 0.0001}	0.0001	0.0001
batch size	[32, 64, 128]	32	64
dropout	[0.0, 0.4, 0.6]	0.6	0.6
w_p	[1, 2, 5, 25]	5	5
N_{neg}	[9, 49, 199]	49	49

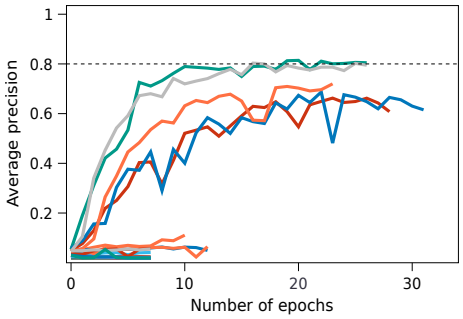


Figure B.1: Validation AP of tuned DeepCorr configurations.

AP over the whole training process. Note that there are several identical configurations among the top 10. This is due to the framework evaluating some of the most promising hyperparameter configurations repeatedly, in order to account for variability of the training process and verify superiority of given parametrizations.

Figure B.2 depicts the complementing part of Figure 4.6 in Section 4.6, with the validation loss instead of AP. Tables B.3 and B.4 provide information on all individual evaluation runs of respectively $\mathcal{TDC}_{\mathcal{SD}}$ and $\mathcal{TDC}_{\mathcal{MD}}$ on five folds of each dataset over four varied random seeds. These results illustrate high variability of deep neural networks (even on the *exactly same* data and configurations), thus underlining the importance of Algorithm 4.1.

Table B.2: Top performing hyperparameter tuning runs for the $\mathcal{TDC}$ model on a subset of the single-proxy dataset SD (top) and multi-proxy dataset MD (bottom). Dropout = 0.6. Sorted by the final AP. The best configuration is chosen based on the maximum achieved AP, marked in bold.

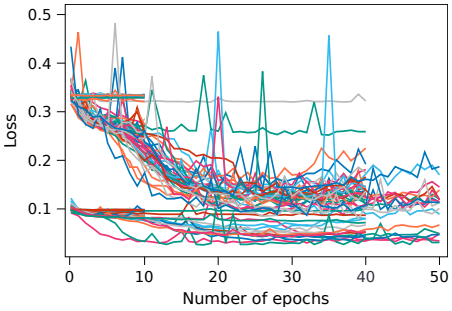
	lr	batch size	w_p	dense layers	kernels	loss	final AP
SD	0.001	128	1	[500, 100]	[128, 128]	0.031	0.821
	0.0001	**32**	**5**	**[500, 100]**	**[64, 128]**	0.089	0.748
	—"—	32	1	—"—	—"—	0.034	0.747
	—"—	64	5	—"—	—"—	0.114	0.744
	—"—	64	1	—"—	—"—	0.047	0.740

	lr	batch size	w_p	dense layers	kernels	loss	final AP
MD	**0.0001**	**64**	**5**	**[500,100]**	**[64, 128]**	0.184	0.667
	—"—	—"—	5	—"—	[128, 128]	0.192	0.664
	—"—	—"—	1	—"—	[64, 128]	0.065	0.637
	—"—	—"—	1	—"—	—"—	0.061	0.635
	—"—	—"—	5	—"—	—"—	0.178	0.608

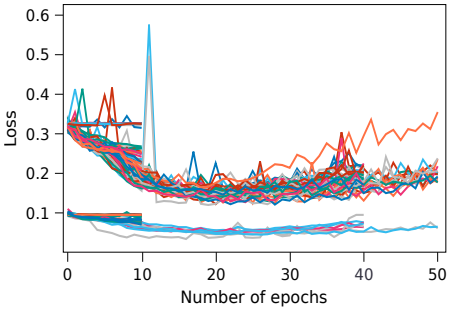
B.3 Comparison of Attacks

With Algorithm B.1, we present our proposal for replication and scientific comparison of DL-based attacks against Tor, including E2E traffic correlation and website fingerprinting. The main intended usage is a systematic evaluation of novel attacks against the state-of-the-art. This algorithm follows the general logic of Algorithm 4.1 and has similar motivation. Namely, in order to aim for reliable conclusions on superiority of one data-driven approach over the other based on limited empirical evidence, it is essential to account for major sources of potential bias and variability. While cross-validation is usually used to diminish possible sampling bias, other aspects are not very often considered in the literature: (1) re-tuning on data that was not used for the original tuning of the state-of-the-art attack; (2) coordination of tuning processes for both attacks in terms of time and memory resources, available search spaces, and conditions and metrics for early stopping; (3) randomization in training of neural networks, which is manifested e. g. in random weight initialization and random data shuffling during stochastic gradient descent. And finally, the algorithm suggests a Welch t-test for statistical significance in case of unequal standard deviations.[1]

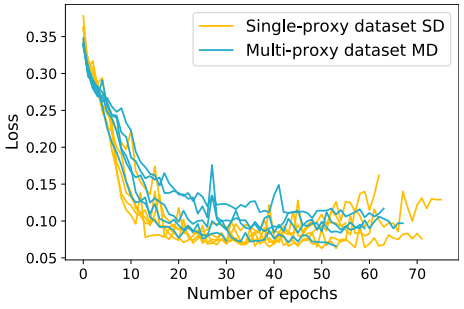
[1] Same as in the original algorithm, we do not highlight what is the most suitable choice of a significance test for the given problem, but instead use a Welch test that assumes normal distribution.



(a) Validation loss of tested configurations on single-proxy dataset SD.



(b) Validation loss of tested configurations on multi-proxy dataset MD.



(c) Validation loss during training.

Figure B.2: Hyperparameter tuning of $\mathcal{TDC}$. Subfigures (a) and (b) depict validation loss of top-120 configurations over 50 epochs on the SD and MD datasets, respectively. Note that runs are terminated early when they do not progress well enough. Subfigure (c) depicts cross-validation of $\mathcal{TDC}$ on both datasets with seed 10.

As already discussed in Section 4.5, exhaustive evaluations are unattainable. The ideal evaluation may become too computationally intensive and tedious to perform, especially with large datasets and models. This often prompts researchers to compromise on some aspects of a thorough evaluation, to obtain a desirable cost/benefit trade-off in terms of computational costs and reliability of results. In such situations, we encourage to explicitly acknowledge which parts of the algorithm have been omitted or reduced for a computationally feasible comparison. For example, compared attacks might have had inconsistent tuning conditions, or performance samples of one attack were collected over a bigger range of random seeds because of a higher variance. Reporting these experimental limitations along with the results is instrumental for transparency and for estimation of certainty in a given outcome.

Table B.3: Detailed attack results for single-proxy model $\mathcal{TDC}_{SD}$ with cross-validation and varied seeds.

$\mathcal{TDC}_{SD}$				$FPR = 0.001$		$TPR = 80\,\%$	
	Test fold	$AP_{val}(\%)$	$AP_{test}(\%)$	$TPR(\%)$	$Pr(\%)$	FPR	$Pr(\%)$
Seed = 1	■□□□□	77.67	26.08	48.55	21.83	0.0066	7.05
	□■□□□	77.34	17.98	38.24	18.04	0.0091	5.19
	□□■□□	86.13	27.24	53.02	23.29	0.0046	9.71
	□□□■□	64.87	11.79	31.82	15.42	0.0135	3.57
	□□□□■	81.3	33.27	55.97	24.3	0.0051	8.9
Seed = 10	■□□□□	84.69	29.09	55.47	24.14	0.0039	11.34
	□■□□□	82.57	24.46	47.74	21.51	0.0059	7.75
	□□■□□	85.99	27.99	54.4	23.91	0.0046	9.68
	□□□■□	83.44	34.17	59.12	25.28	0.0036	12.01
	□□□□■	84.65	29.3	55.28	24.1	0.0048	9.48
Seed = 101	■□□□□	67.49	14.38	33.9	16.31	0.0115	4.16
	□■□□□	81.2	29.95	53.52	23.65	0.0054	8.4
	□□■□□	84.76	26.04	52.08	22.99	0.0047	9.56
	□□□■□	71.83	16.05	36.86	17.47	0.0114	4.19
	□□□□■	76.34	17.51	39.87	18.59	0.0109	4.37
Seed = 102	■□□□□	66.05	13.01	31.76	15.47	0.0143	3.36
	□■□□□	83.69	31.74	55.03	23.98	0.0047	9.54
	□□■□□	84.01	27.61	49.87	22.39	0.0059	7.82
	□□□■□	84.04	30.34	56.16	24.35	0.0037	11.86
	□□□□■	76.43	23.1	46.6	21.06	0.0081	5.8
	Mean $\pm$ **std**	**79.22** $\pm$ **6.80**	**24.55** $\pm$ **6.98**	**47.76** $\pm$ **9.01**	**21.40** $\pm$ **3.27**	**0.0071** $\pm$ **0.003**	**7.68** $\pm$ **2.82**

B.4 Explainability Graphs

In Section 4.6, we perform explainability analysis to test whether certain timing characteristics are more influential for one data collection setup than the other. Here we give more detailed results of the explainability experiments, for models created with random seeds 1 and 10 and trained on all 5 folds. This experiment provides preliminary ideas on the presence of potential timing bias that may be simplifying traffic correlation in the single-proxy setup. Figure B.3 depicts differences in mean activations between truly-associated and non-associated traces. Since the visual analysis is challenging, Table B.5 contains total (summed up) differences between these activations maps for both uplink and downlink timings. We observe that uplink timings mostly have much larger impact on single-proxy models than on multi-proxy models, except two scenarios. This preliminary result could be an indication of subtle timing bias being present in single-proxy uplink timings. To obtain conclusive results, we propose extending this analysis beyond analysis of activation maps.

Table B.4: Detailed attack results for multi-proxy model $\mathcal{TDC}_{MD}$ with cross-validation and varied seeds. Two outlier runs (seed=101, fold=2,4) marked red were excluded from computation of mean and statistical significance.

$\mathcal{TDC}_{MD}$				$FPR = 0.001$		$TPR = 80\,\%$	
	Test fold	$AP_{val}(\%)$	$AP_{test}(\%)$	$TPR(\%)$	$Pr(\%)$	FPR	$Pr(\%)$
Seed = 1	■□□□□	61.27	6.73	20.38	10.45	0.0281	1.74
	□■□□□	60.2	6.21	19.87	10.24	0.0246	1.98
	□□■□□	62.2	9.05	26.04	13.07	0.0196	2.48
	□□□■□	84.75	24.23	47.92	21.62	0.0063	7.27
	□□□□■	75.38	19.37	40.94	18.99	0.0091	5.18
Seed = 10	■□□□□	81.85	14.41	35.03	16.73	0.01	4.74
	□■□□□	75.43	15.83	34.84	16.65	0.0104	4.56
	□□■□□	78.44	21.45	46.42	21.01	0.0065	7.12
	□□□■□	65.13	7.7	22.58	11.44	0.0237	2.06
	□□□□■	78.75	16.59	37.99	17.86	0.009	5.25
Seed = 101	■□□□□	79.44	18.33	41.13	19.11	0.0084	5.58
	□■□□□	22.35	0.96	4.97	2.78	0.1313	0.38
	□□■□□	51.77	4.84	15.66	8.35	0.0343	1.43
	□□□■□	13.19	0.53	3.02	1.71	0.2103	0.24
	□□□□■	76.96	18.37	37.99	17.93	0.0101	4.71
Seed = 102	■□□□□	60.97	6.26	21.57	11.1	0.0243	2
	□■□□□	68.64	9.81	26.92	13.35	0.0173	2.8
	□□■□□	60.83	6.93	21.19	10.83	0.0216	2.25
	□□□■□	80.4	19.8	42.83	19.7	0.0077	6.04
	□□□□■	80.51	23.03	44.47	20.28	0.0074	6.32
	Mean	**71.27**	**13.83**	**32.43**	**15.02**	**0.0154**	**4.08**
	± std	**± 9.84**	**± 6.60**	**± 10.59**	**± 5.19**	**± 0.008**	**± 1.98**

Algorithm B.1: Replication and comparison of two ML/DL attacks on the same dataset. Here, with AP as a target metric and t-test for statistical significance.

Input: Dataset D; ML/DL attacks A_1, A_2; hypeparameter search spaces S_{A_1}, S_{A_2}.
Output: Attack with the highest mean performance on D; diff between means; statistical significance.

Set number of cross-validation folds K
Set maximum numbers of attack configurations C_{A_1}, C_{A_2} (or consistent time and memory resources)
Set number of randomized training runs n
Split D into K equal folds $D_{1,...,K}$
$AP_{A_1} \leftarrow \{\}$, $AP_{A_2} \leftarrow \{\}$
for $k \in \{1, ..., K\}$ **do**
    Put aside fold D_k as $test_k$
    Assign the remaining folds to $train_k$
    Sample $tune_k \subseteq train_k$
    for *attack* $A \in \{A_1, A_2\}$ **do**
        $AP_{best} \leftarrow 0$
        for $i \in \{1, ..., C_A\}$ **do**
            Sample config c_i from S_A using the chosen optimization strategy
            Split $tune_k$ into $tune_{train}$, $tune_{val}$
            Train A with config c_i on $tune_{train}$
            Compute AP on $tune_{val}$
            if $AP > AP_{best}$ **then**
                $c_k = c_i$, $AP_{best} = AP$
        repeat n **times**
            Change random seed
            Train A with config c_k on $train_k$
            Compute AP on $test_k$
            $AP_A \leftarrow AP_A \cup AP$
Compute t-test and p-value on (AP_{A_1}, AP_{A_2})
return *argmax*$(\overline{AP}_{A_1}, \overline{AP}_{A_2})$; $|\overline{AP}_{A_1} - \overline{AP}_{A_2}|$; *p-value.*

Table B.5: Comparison of the impact of uplink timings vs. downlink timings on the traffic correlation attack by single- and multi-proxy models created with given seeds and folds. For both models, we first show the total difference in mean activations for uplink timings (A_{T^u}), as well as for downlink timings (A_{T^d}), followed by their subtraction ($A_{T^u} - A_{T^d}$). Positive results of subtraction show that uplink values are always larger. Then, the delta column shows how much larger the relative impact of uplink timings is on single-proxy models than it is on multi-proxy models. For reference, the right side of the table lists the corresponding attack performance metrics.

		$\mathcal{TDC}_{SD}$			$\mathcal{TDC}_{MD}$				$\mathcal{TDC}_{SD}$		$\mathcal{TDC}_{MD}$	
	Test fold	A_{T^u}	A_{T^d}	A_{T^u}-A_{T^d}	A_{T^u}	A_{T^d}	A_{T^u}-A_{T^d}	Delta	AP_{val} %	AP_{test} %	AP_{val} %	AP_{test} %
Seed = 10	■□□□□	226720.5	101278.8	125441.7	247898.9	179573.3	68325.5	46 %	84.69	29.09	81.85	14.41
	□■□□□	226106.2	112463.7	113642.6	148192.4	76552.0	71640.4	37 %	82.57	24.46	75.43	15.83
	□□■□□	183858.9	85385.1	98473.8	167102.9	113232.8	53870.1	45 %	85.99	27.99	78.44	21.45
	□□□■□	207299.5	80356.2	126943.3	200730.1	124901.6	75828.4	40 %	83.44	34.17	65.13	7.7
	□□□□■	207509.1	90500.0	117009.1	188045.7	146269.1	41776.6	64 %	84.65	29.3	78.75	16.59
Seed = 1	■□□□□	180377.9	98670.1	81707.8	246522.6	154175.9	92346.7	−13 %	77.67	26.08	61.27	6.73
	□■□□□	193831.3	92423.1	101408.2	213124.6	100228.8	112895.8	−11 %	77.34	17.98	60.2	6.21
	□□■□□	177800.0	86152.4	91647.7	155957.5	102699.1	53258.3	42 %	86.13	27.24	62.2	9.05
	□□□■□	191768.9	79840.0	111928.9	190277.7	117061.6	73216.1	35 %	64.87	11.79	84.75	24.23
	□□□□■	170874.1	76537.1	94337.0	176942.0	151082.6	25859.5	73 %	81.3	33.27	75.38	19.37

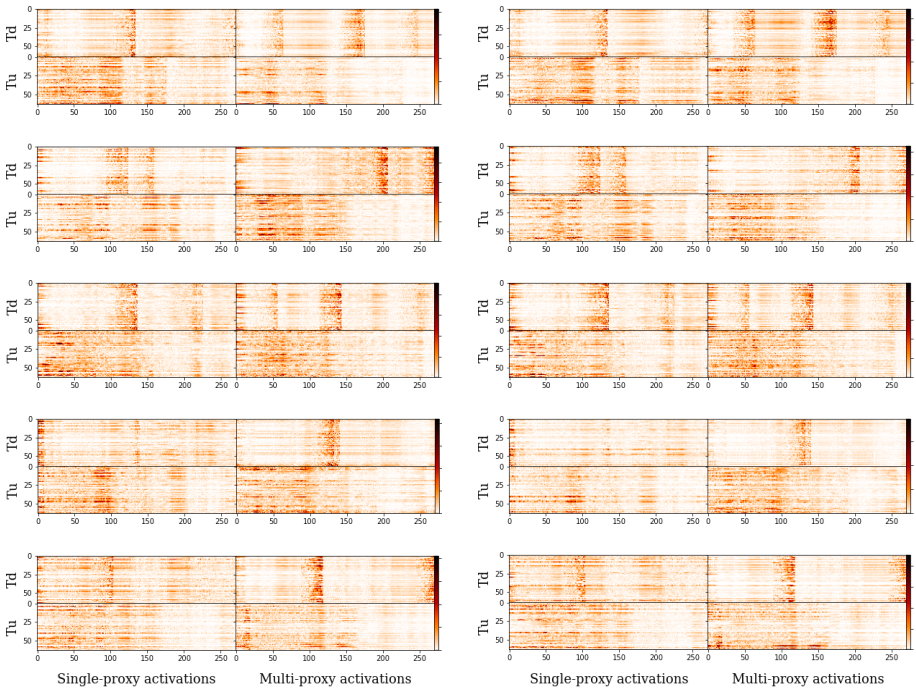


Figure B.3: Differences between mean activation maps computed over 1590 truly-associated (pos) and non-associated (neg) traces. Individual heatmaps display on the horizontal axis single-proxy (left) vs. multi-proxy (right), and on the vertical axis differences between mean activations generated from uplink timings (T^u) vs. downlink (T^d). Darker color indicates higher intensity. All maps in the left column were produced with seed 1 vs. seed 10 for those in the right column. Folds 1 to 5 are arranged from top to bottom.

Appendix C

Additional Materials for Deep Learning for NIDS

C.1 Hyperparameter Tuning Details

This appendix contains the details of sequential deep learning models we design and evaluate in Chapter 6, for both misuse and anomaly detection. Table C.1 presents hyperparameter tuning details for deep learning-based NIDS.

Table C.1: Model selection and hyperparameter tuning details for deep learning architectures used for network intrusion detection on raw traffic.

Hyperparameter	Search space	Value	Search space	Value
	LSTM		BiLSTM	
units	64, 128, 256	128	32, 64, 128	128
activation	tanh, relu	relu	tanh, relu	relu
dense units	64+64, 128	64+64	64+64, 128	64+64
bi-direction	—	—	concat, sum	sum
optimizer	SGD, Adam	Adam	SGD, Adam	Adam
learning rate	0.0008 ... 0.0012	0.001	0.0008 ... 0.001	0.001
batch size	64, 128, 256	128	64, 128, 256, 512	256
dropout	0.0...0.2	0.0	0.0...0.2	0.0
	TCN		TCN AE	
layers	6-20	12	6-20	17
kernels	32, 64	32	8, 64, 128	128
kernel size	2, 3, 5	3	2, 3	2
padding	—	causal	—	causal
dilations	—	1 – 32	—	1 – 32
activation	tanh, relu	relu	tanh, relu	relu
optimizer	SGD, Adam	Adam	—	Adam
learning rate	0.0008 ... 0.0012	0.001	0.0004 ... 0.001	0.001
batch size	64, 128, 256	64	256, 512	512
dropout	0.0-0.2	0.2	0.0-0.2	0.0
	LSTM-Att AE			
units	10, 32, 64	10		
layers	1, 2	1		
optimizer	—	Adam		
learning rate	0.0008 ... 0.002	0.0009		
batch size	128, 256, 256	512		
dropout	0.0-0.2	0.0		

Bibliography

[1] K. Abe and S. Goto. "Fingerprinting Attack on Tor Anonymity using Deep Learning". In: *Proceedings of the Asia-Pacific Advanced Network* 42, 2016, pp. 15–20.

[2] G. Acar, M. Juarez, and individual contributors. *tor-browser-selenium: Tor Browser Automation with Selenium*. 2020. URL: `https://github.com/webfp/tor-browser-selenium`.

[3] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé. "Mobile Encrypted Traffic Classification using Deep Learning". In: *2018 Network traffic measurement and analysis conference (TMA)*. IEEE. 2018, pp. 1–8.

[4] M. Ahmed, A. N. Mahmood, and J. Hu. "A Survey of Network Anomaly Detection Techniques". In: *Journal of Network and Computer Applications* 60, 2016, pp. 19–31.

[5] M. Alber, S. Lapuschkin, P. Seegerer, M. Hägele, K. T. Schütt, G. Montavon, W. Samek, K.-R. Müller, S. Dähne, and P.-J. Kindermans. "iNNvestigate Neural Networks!" In: *J. Mach. Learn. Res.* 20.93, 2019, pp. 1–8.

[6] E. Albin and N. C. Rowe. "A Realistic Experimental Comparison of the Suricata and Snort Intrusion-Detection Systems". In: *2012 26th International Conference on Advanced Information Networking and Applications Workshops*. IEEE. 2012, pp. 122–127.

[7] K. Alrawashdeh and C. Purdy. "Toward an Online Anomaly Intrusion Detection System based on Deep Learning". In: *15th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2016, pp. 195–200.

[8] K. Amarasinghe, K. Kenney, and M. Manic. "Toward Explainable Deep Neural Network based Anomaly Detection". In: *2018 11th International Conference on Human System Interaction (HSI)*. IEEE. 2018, pp. 311–317.

[9] B. Anderson and D. McGrew. “Machine Learning for Encrypted Malware Traffic Classification: Accounting for Noisy Labels and Non-Stationarity”. In: *Proceedings of the 23rd ACM SIGKDD International Conference on knowledge discovery and data mining*. 2017, pp. 1723–1732.

[10] G. Andresini, F. Pendlebury, F. Pierazzi, C. Loglisci, A. Appice, and L. Cavallaro. “Insomnia: Towards Concept-Drift Robustness in Network Intrusion Detection”. In: *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*. 2021, pp. 111–122.

[11] G. Apruzzese and M. Colajanni. “Evading Botnet Detectors Based on Flows and Random Forest with Adversarial Samples”. In: *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. IEEE. 2018, pp. 1–8.

[12] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz. “Invariant Risk Minimization”. In: *arXiv preprint arXiv:1907.02893*, 2019.

[13] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck. “Dos and Don’ts of Machine Learning in Computer Security”. In: *USENIX Security Symposium*. 2022.

[14] D. Arpit, S. Jastrzębski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio, et al. “A Closer Look at Memorization in Deep Networks”. In: *International conference on machine learning*. PMLR. 2017, pp. 233–242.

[15] S. Axelsson. “The Base-Rate Fallacy and Its Implications for the Difficulty of Intrusion Detection”. In: *Proceedings of the 6th ACM Conference on Computer and Communications Security*. 1999, pp. 1–7.

[16] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. “On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation”. In: *PloS one* 10.7, 2015, e0130140.

[17] V. Badrinarayanan, A. Kendall, and R. Cipolla. “Segnet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.12, 2017, pp. 2481–2495.

[18] S. Bai, J. Zico Kolter, and V. Koltun. *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. Tech. rep. arXiv: 1803.01271v2. URL: http://github.com/locuslab/TCN..

[19] M. Basseville, I. V. Nikiforov, et al. *Detection of Abrupt Changes: Theory and Application*. Vol. 104. prentice Hall Englewood Cliffs, 1993.

[20] A. Bendale and T. Boult. “Towards Open World Recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1893–1902.

[21] J. Bergstra, D. Yamins, and D. Cox. “Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures”. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by S. Dasgupta and D. McAllester. Vol. 28. 1. PMLR, June 2013, pp. 115–123.

[22] D. S. Berman, A. L. Buczak, J. S. Chavis, and C. L. Corbett. “A Survey of Deep Learning Methods for Cyber Security”. In: *Information* 10.4, 2019, p. 122.

[23] M. Bermúdez-Edo, R. Salazar-Hernández, J. Díaz-Verdejo, and P. Garcia-Teodoro. “Proposals on assessment environments for anomaly-based network intrusion detection systems”. In: *International Workshop on Critical Information Infrastructures Security*. Springer. 2006, pp. 210–221.

[24] S. Bhat, D. Lu, A. Kwon, and S. Devadas. “Var-cnn: A Data-Efficient Website Fingerprinting Attack Based On Deep Learning”. In: *Privacy Enhancing Technologies Symposium* 4, 2019, pp. 292–310.

[25] S. Bhatt, P. K. Manadhata, and L. Zomlot. “The Operational Role of Security Information and Event Management Systems”. In: *IEEE security & Privacy* 12.5, 2014, pp. 35–41.

[26] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine. “Privacy Vulnerabilities in Encrypted HTTP Streams”. In: *Workshop on Privacy Enhancing Technologies*. PET ’05. Springer. May 2005, pp. 1–11.

[27] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo. “A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities”. In: *Journal of Internet Services and Applications* 9.1, 2018, pp. 1–99.

[28] K. Boyd, V. Santos Costa, J. Davis, and C. D. Page. “Unachievable Region in Precision-Recall Space and Its Effect on Empirical Evaluation.” In: *Proceedings of the 29th International Conference on Machine Learning (ICML)*. Vol. 2012. 2012, p. 349.

[29] R. A. Bridges, T. R. Glass-Vanderlan, M. D. Iannacone, M. S. Vincent, and Q. Chen. “A Survey of Intrusion Detection Systems Leveraging Host Data”. In: *ACM Computing Surveys (CSUR)* 52.6, 2019, pp. 1–35.

[30] J. Buchner. *ImageHash*. 2017. URL: `https://github.com/JohannesBuchner/imagehash`.

[31] A. L. Buczak and E. Guven. “A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection”. In: *IEEE Communications surveys & tutorials* 18.2, 2015, pp. 1153–1176.

[32] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg. "A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses". In: *ACM Conference on Computer and Communications Security*. ACM, 2014, pp. 227–238.

[33] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. "Touching from a Distance: Website Fingerprinting Attacks and Defenses". In: *ACM Conference on Computer and Communications Security*. ACM, Oct. 2012, pp. 605–616.

[34] N. Carlini and D. Wagner. "Towards Evaluating the Robustness of Neural Networks". In: *IEEE European Symposium on Security and Privacy*. EuroS&P '17. 2017, pp. 39–57.

[35] P. Casas, F. Soro, J. Vanerio, G. Settanni, and A. D'Alconzo. "Network Security and Anomaly Detection with Big-DAMA: A Big Data Analytics Framework". In: *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*. IEEE. 2017, pp. 1–7.

[36] H.-s. Chae, B.-o. Jo, S.-H. Choi, and T.-k. Park. "Feature Selection for Intrusion Detection using NSL-KDD". In: *Recent advances in computer science* 20132, 2013, pp. 184–187.

[37] V. Chandola, A. Banerjee, and V. Kumar. "Anomaly Detection: A Survey". In: *ACM computing surveys (CSUR)* 41.3, 2009, pp. 1–58.

[38] M. S. Charikar. "Similarity Estimation Techniques from Rounding Algorithms". In: *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM. 2002, pp. 380–388.

[39] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. "Return of the Devil in the Details: Delving Deep into Convolutional Nets". In: *arXiv preprint arXiv:1405.3531*, 2014.

[40] H. Cheng and R. Avnur. "Traffic Analysis of SSL Encrypted Web Browsing". In: *Project paper, University of Berkeley*, 1998.

[41] G. Cherubin, J. Hayes, and M. Juarez. "Website Fingerprinting Defenses at the Application Layer". In: *Privacy Enhancing Technologies Symposium*. De Gruyter, 2017, pp. 168–185.

[42] G. Cherubin, R. Jansen, and C. Troncoso. "Online Website Fingerprinting: Evaluating Website Fingerprinting Attacks on Tor in the Real World". In: *USENIX Security Symposium*. 2022, pp. 753–770.

[43] F. Chollet et al. *Keras*. 2015. URL: `https://github.com/fchollet/keras`.

[44] B. Claise, G. Sadasivan, V. Valluri, and M. Djernaes. "Cisco Systems Netflow Services Export Version 9". In: 2004.

[45] H. Clausen and D. Aspinall. “Examining Traffic Microstructures to Improve Model Development”. In: *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2021, pp. 19–24.

[46] H. Clausen, R. Flood, and D. Aspinall. “Controlling Network Traffic Microstructures for Machine-Learning Model Probing”. In: *17th EAI International Conference on Security and Privacy in Communication Networks*. 2021.

[47] I. Corona, G. Giacinto, and F. Roli. “Adversarial Attacks against Intrusion Detection Systems: Taxonomy, Solutions and Open Issues”. In: *Information Sciences* 239, 2013, pp. 201–225.

[48] G. F. Cretu, A. Stavrou, M. E. Locasto, S. J. Stolfo, and A. D. Keromytis. “Casting Out Demons: Sanitizing Training Data for Anomaly Sensors”. In: *2008 IEEE Symposium on Security and Privacy (sp 2008)*. IEEE. 2008, pp. 81–95.

[49] S. d’Ascoli, L. Sagun, G. Biroli, and J. Bruna. “Finding the Needle in the Haystack with Convolutions: On the Benefits of Architectural Bias”. In: *Advances in Neural Information Processing Systems* 32, 2019.

[50] M. F. Dacrema, P. Cremonesi, and D. Jannach. “Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches”. In: *Proceedings of the 13th ACM Conference on Recommender Systems*. 2019, pp. 101–109.

[51] T. Dahanayaka, G. Jourjon, and S. Seneviratne. “Dissecting Traffic Fingerprinting CNNs with Filter Activations”. In: *Computer Networks*, 2022, p. 108770.

[52] G. Danezis. “The traffic analysis of continuous-time mixes”. In: *Workshop on Privacy Enhancing Technologies*. PET ’04. Springer, May 2004, pp. 35–50.

[53] J. J. Davis and A. J. Clark. “Data Preprocessing for Anomaly Based Network Intrusion Detection: A Review”. In: *computers & security* 30.6-7, 2011, pp. 353–375.

[54] W. De la Cadena, A. Mitseva, J. Hiller, J. Pennekamp, S. Reuter, J. Filter, T. Engel, K. Wehrle, and A. Panchenko. “Trafficsliver: Fighting website fingerprinting attacks with traffic splitting”. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 1971–1985.

[55] G. De Palma, B. T. Kiani, and S. Lloyd. “Deep Neural Networks are Biased Towards Simple Functions”. In: *arXiv* 2018, 2018.

[56] J. Demšar. “Statistical Comparisons of Classifiers over Multiple Data Sets”. In: *The Journal of Machine Learning Research* 7, 2006, pp. 1–30.

[57] R. Dingledine, N. Mathewson, and P. F. Syverson. “Tor: The Second-Generation Onion Router”. In: *USENIX Security Symposium*. USENIX Association, 2004, pp. 303–320.

[58] Y. Dong, Y. Zhang, H. Ma, Q. Wu, Q. Liu, K. Wang, and W. Wang. “An Adaptive System for Detecting Malicious Queries in Web Attacks”. In: *Science China Information Sciences* 61.3, 2018, p. 032114.

[59] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martinez-del-Rincon, and D. Siracusa. “LUCID: A practical, Lightweight Deep Learning Solution for DDoS Attack Detection”. In: *IEEE Transactions on Network and Service Management* 17.2, 2020, pp. 876–889.

[60] R. Dror, L. Peled-Cohen, S. Shlomov, and R. Reichart. “Statistical Significance Testing for Natural Language Processing”. In: *Synthesis Lectures on Human Language Technologies* 13.2, 2020, pp. 1–116.

[61] M. Du, N. Liu, and X. Hu. “Techniques for Interpretable Machine Learning”. In: *Communications of the ACM* 63.1, 2019, pp. 68–77.

[62] M. Du, F. Li, G. Zheng, and V. Srikumar. “Deeplog: Anomaly Detection and Diagnosis from System Logs through Deep Learning”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 1285–1298.

[63] Z. Durumeric, M. Bailey, and J. A. Halderman. “An Internet-Wide View of Internet-Wide Scanning”. In: *USENIX Security Symposium*. 2014, pp. 65–78.

[64] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. “Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail”. In: *IEEE Symposium on Security and Privacy*. S&P ’12. IEEE, 2012, pp. 332–346.

[65] G. Engelen, V. Rimmer, and W. Joosen. “Troubleshooting an Intrusion Detection Dataset: The CICIDS2017 Case Study”. In: *2021 IEEE Security and Privacy Workshops (SPW)*. May 2021, pp. 7–12.

[66] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie. “High-Dimensional and Large-Scale Anomaly Detection using a Linear One-Class SVM with Deep Learning”. In: *Pattern Recognition* 58, 2016, pp. 121–134.

[67] S. Eun Oh, T. Yang, N. Mathews, J. K. Holland, M. S. Rahman, M. Wright, and N. Hopper. “DeepCoFFEA: Improved Flow Correlation Attacks on Tor via Metric Learning and Amplification”. In: *IEEE Symposium on Security and Privacy*. S&P ’22. May 2022.

[68] W. Feng, Q. Zhang, G. Hu, and J. X. Huang. "Mining Network Data for Intrusion Detection through Combining SVMs with Ant Colony Networks". In: *Future Generation Computer Systems* 37, 2014, pp. 127–140.

[69] G. Fernandes, J. J. Rodrigues, L. F. Carvalho, J. F. Al-Muhtadi, and M. L. Proença. "A Comprehensive Survey on Network Anomaly Detection". In: *Telecommunication Systems* 70.3, 2019, pp. 447–489.

[70] P. Fogla, M. I. Sharif, R. Perdisci, O. M. Kolesnikov, and W. Lee. "Polymorphic Blending Attacks". In: *USENIX Security Symposium*. 2006, pp. 241–256.

[71] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda. "MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking". In: *ACM CoNEXT '10*. Dec. 2010.

[72] G. M. D. T. Forecast. "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017–2022". In: *Update* 2017, 2019, p. 2022.

[73] N. Gao, L. Gao, Q. Gao, and H. Wang. "An Intrusion Detection Model based on Deep Belief Networks". In: *2014 Second International Conference on Advanced Cloud and Big Data*. IEEE. 2014, pp. 247–252.

[74] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. "Anomaly-Based Network Intrusion Detection: Techniques, Systems and Challenges". In: *computers & security* 28.1-2, 2009, pp. 18–28.

[75] G. Garofalo, V. Rimmer, T. Van hamme, D. Preuveneers, W. Joosen, et al. "Fishy Faces: Crafting Adversarial Images to Poison Face Authentication". In: *USENIX Workshop on Offensive Technologies*. WOOT '18. Aug. 2018, pp. 1–12.

[76] C. Gates and C. Taylor. "Challenging the Anomaly Detection Paradigm: A Provocative Discussion". In: *Proceedings of the 2006 workshop on New security paradigms*. 2006, pp. 21–29.

[77] R. Geirhos, J.-H. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge, and F. A. Wichmann. "Shortcut Learning in Deep Neural Networks". In: *Nature Machine Intelligence* 2.11, 2020, pp. 665–673.

[78] J. Gong and T. Wang. "Zero-Delay Lightweight Defenses Against Website Fingerprinting". In: *USENIX Security Symposium*. USENIX Association, 2020, pp. 717–734.

[79] M. Grill and T. Pevný. "Learning Combination of Anomaly Detectors for Security Domain". In: *Computer Networks* 107, 2016, pp. 55–63.

[80] M. Grill, T. Pevný, and M. Rehak. "Reducing False Positives of Network Anomaly Detection by Local Adaptive Multivariate Smoothing". In: *Journal of Computer and System Sciences* 83.1, 2017, pp. 43–57.

[81] C. Guo, Y. Ping, N. Liu, and S.-S. Luo. “A Two-Level Hybrid Approach for Intrusion Detection”. In: *Neurocomputing* 214, 2016, pp. 391–400.

[82] C. Guo, Y.-J. Zhou, Y. Ping, S.-S. Luo, Y.-P. Lai, and Z.-K. Zhang. “Efficient Intrusion Detection using Representative Instances”. In: *computers & security* 39, 2013, pp. 255–267.

[83] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing. “Lemna: Explaining Deep Learning Based Security Applications”. In: *proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. 2018, pp. 364–379.

[84] R. A. A. Habeeb, F. Nasaruddin, A. Gani, I. A. T. Hashem, E. Ahmed, and M. Imran. “Real-Time Big Data Processing for Anomaly Detection: A Survey”. In: *International Journal of Information Management* 45, 2019, pp. 289–307.

[85] J. Hayes and G. Danezis. “Guard Sets for Onion Routing”. In: vol. 2015. PoPETS ’15 2. De Gruyter, 2015, pp. 65–80.

[86] J. Hayes and G. Danezis. “k-Fingerprinting: A Robust Scalable Website Fingerprinting Technique”. In: *USENIX Security Symposium*. USENIX ’16. USENIX Association, Aug. 2016, pp. 1187–1203.

[87] F. E. Heba, A. Darwish, A. E. Hassanien, and A. Abraham. “Principle Components Analysis and Support Vector Machine based Intrusion Detection System”. In: *2010 10th international conference on intelligent systems design and applications*. IEEE. 2010, pp. 363–367.

[88] C. Heinze-Deml and N. Meinshausen. “Conditional Variance Penalties and Domain Shift Robustness”. In: *Machine Learning* 110.2, 2021, pp. 303–348.

[89] D. Herrmann, R. Wendolsky, and H. Federrath. “Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier”. In: *ACM Workshop on Cloud Computing Security*. CCSW ’09. ACM, Nov. 2009, pp. 31–42.

[90] D. Herrmann, R. Wendolsky, and H. Federrath. “Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier”. In: *ACM Workshop on Cloud Computing Security*. ACM, 2009, pp. 31–42.

[91] G. E. Hinton, S. Osindero, and Y.-W. Teh. “A Fast Learning Algorithm for Deep Belief Nets”. In: *Neural computation* 18.7, 2006, pp. 1527–1554.

[92] A. Hintz. “Fingerprinting Websites Using Traffic Analysis”. In: *Privacy Enhancing Technologies Symposium*. Springer, 2003, pp. 171–178.

[93] S. Hochreiter and J. Schmidhuber. “Long Short-Term Memory”. In: *Neural computation* 9.8, 1997, pp. 1735–1780.

[94] J. Holland, P. Schmitt, N. Feamster, and P. Mittal. "New Directions in Automated Traffic Analysis". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 3366–3383.

[95] D. Hoplaros, Z. Tari, and I. Khalil. "Data Summarization for Network Traffic Monitoring". In: *Journal of network and computer applications* 37, 2014, pp. 194–205.

[96] K. Hornik, M. Stinchcombe, and H. White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5, 1989, pp. 359–366.

[97] F. Iglesias, A. Hartl, T. Zseby, and A. Zimek. "Are Network Attacks Outliers? A Study of Space Representations and Unsupervised Algorithms". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2019, pp. 159–175.

[98] A. S. Jacobs, R. Beltiukov, W. Willinger, R. A. Ferreira, A. Gupta, and L. Z. Granville. "AI/ML for Network Security: The Emperor has no Clothes". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2022, pp. 1537–1551.

[99] J.-H. Jacobsen, J. Behrmann, R. Zemel, and M. Bethge. "Excessive Invariance Causes Adversarial Vulnerability". In: *arXiv preprint arXiv:1811.00401*, 2018.

[100] A. D. Jaggard and P. Syverson. "Onions in the Crosshairs: When The Man really is out to get you". In: *Workshop on Privacy in the Electronic Society*. WPES '17. ACM, 2017, pp. 141–151.

[101] R. Jansen and N. Hopper. "Shadow: Running Tor in a Box for Accurate and Efficient Experimentation". In: *Network and Distributed System Security Symposium*. NDSS '12. The Internet Society, Feb. 2012.

[102] W. H. Jefferys and J. O. Berger. "Ockham's Razor and Bayesian Analysis". In: *American Scientist* 80.1, 1992, pp. 64–72.

[103] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson. "Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries". In: *ACM Conference on Computer and Communications Security*. CCS '13. ACM, Nov. 2013, pp. 337–348.

[104] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt. "A Critical Evaluation of Website Fingerprinting Attacks". In: *ACM Conference on Computer and Communications Security*. CCS '14. ACM, Nov. 2014, pp. 263–274.

[105] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright. "Toward an Efficient Website Fingerprinting Defense". In: *European Symposium on Research in Computer Security*. Springer, 2016, pp. 27–46.

[106] A. Kantchelian, S. Afroz, L. Huang, A. C. Islam, B. Miller, M. C. Tschantz, R. Greenstadt, A. D. Joseph, and J. Tygar. "Approaches to Adversarial Drift". In: *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*. 2013, pp. 99–110.

[107] H. G. Kayacik and A. N. Zincir-Heywood. "Mimicry Attacks Demystified: What Can Attackers do to Evade Detection?" In: *2008 Sixth Annual Conference on Privacy, Security and Trust*. IEEE. 2008, pp. 213–223.

[108] S. S. Khan and M. G. Madden. "One-Class Classification: Taxonomy of Study and Review of Techniques". In: *The Knowledge Engineering Review* 29.3, 2014, pp. 345–374.

[109] G. Kim, S. Lee, and S. Kim. "A Novel Hybrid Intrusion Detection Method Integrating Anomaly Detection with Misuse Detection". In: *Expert Systems with Applications* 41.4, 2014, pp. 1690–1700.

[110] J. Kindervag et al. "Build Security into Your Network's DNA: The Zero Trust Network Architecture". In: *Forrester Research Inc*, 2010, pp. 1–26.

[111] M. Kloft and P. Laskov. "Online Anomaly Detection under Adversarial Impact". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010, pp. 405–412.

[112] C. Kruegel and G. Vigna. "Anomaly Detection of Web-Based Attacks". In: *Proceedings of the 10th ACM conference on Computer and communications security*. 2003, pp. 251–261.

[113] D. C. Le, A. N. Zincir-Heywood, and M. I. Heywood. "Data Analytics on Network Traffic Flows for Botnet Behaviour Detection". In: *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2016, pp. 1–7.

[114] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation". In: *Proceedings of the 26th Annual Network and Distributed System Security Symposium*. NDSS '19. Feb. 2019. DOI: 10.14722/ndss.2019.23386.

[115] Y. LeCun and Y. Bengio. "The Handbook of Brain Theory and Neural Networks". In: ed. by M. A. Arbib. MIT Press, 1998. Chap. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258. ISBN: 0-262-51102-9.

[116] Y. Li, C. Wei, and T. Ma. "Towards Explaining the Regularization Effect of Initial Large Learning Rate in Training Neural Networks". In: *Advances in Neural Information Processing Systems* 32, 2019.

[117] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica. "Tune: A Research Platform for Distributed Model Selection and Training". In: *arXiv preprint arXiv:1807.05118*, 2018.

[118] M. Liberatore and B. N. Levine. “Inferring the source of encrypted HTTP connections”. In: *ACM Conference on Computer and Communications Security*. ACM, 2006, pp. 255–263.

[119] L. Liu, G. Engelen, T. Lynar, D. Essam, and W. Joosen. “Error Prevalence in NIDS datasets: A Case Study on CIC-IDS-2017 and CSE-CIC-IDS-2018”. In: *2022 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2022.

[120] M. Lotfollahi, M. Jafari Siavoshani, R. Shirali Hossein Zade, and M. Saberian. “Deep Packet: A Novel Approach for Encrypted Traffic Classification using Deep Learning”. In: *Soft Computing* 24.3, 2020, pp. 1999–2012.

[121] S. M. Lundberg and S.-I. Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Advances in neural information processing systems* 30, 2017.

[122] M.-T. Luong, H. Pham, and C. D. Manning. “Effective Approaches to Attention-based Neural Machine Translation”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2015, pp. 1412–1421.

[123] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro, and R. Therón. “UGR ’16: A New Dataset for the Evaluation of Cyclostationarity-Based Network IDSs”. In: *Computers & Security* 73, 2018, pp. 411–424.

[124] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. “Towards Deep Learning Models Resistant to Adversarial Attacks”. In: *arXiv preprint arXiv:1706.06083*, 2017.

[125] H. Maghrebi, T. Portigliatti, and E. Prouff. “Breaking Cryptographic Implementations using Deep Learning Techniques”. In: *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer. 2016, pp. 3–26.

[126] M. Mahoney. “Network Traffic Anomaly Detection Based on Packet Bytes”. In: *Proceedings of the 2003 ACM symposium on Applied computing*, p. 346350.

[127] M. Marchetti, F. Pierazzi, M. Colajanni, and A. Guido. “Analysis of High Volumes of Network Traffic for Advanced Persistent Threat Detection”. In: *Computer Networks* 109, 2016, pp. 127–141.

[128] M. Mardani, G. Mateos, and G. B. Giannakis. “Dynamic Anomalography: Tracking Network Anomalies via Sparsity and Low Rank”. In: *IEEE Journal of Selected Topics in Signal Processing* 7.1, 2012, pp. 50–66.

[129] G. Marín, P. Casas, and G. Capdehourat. "Deep in the Dark: Deep Learning-Based Malware Traffic Detection Without Expert Knowledge". In: *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2019, pp. 36–42.

[130] N. Mathews, J. K. Holland, S. E. Oh, M. S. Rahman, N. Hopper, and M. Wright. "SoK: A Critical Evaluation of Efficient Website Fingerprinting Defenses". In: *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society. 2022, pp. 344–361.

[131] N. Mathewson and R. Dingledine. "Practical Traffic Analysis: Extending and Resisting Statistical Disclosure". In: *Workshop on Privacy Enhancing Technologies*. PET '04. Springer, May 2004, pp. 17–34.

[132] R. A. Maxion and K. M. Tan. "Benchmarking Anomaly-Based Detection Systems". In: *Proceeding International Conference on Dependable Systems and Networks. DSN 2000*. IEEE. 2000, pp. 623–630.

[133] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai. "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection". In: *machine learning* 5, p. 2.

[134] Y. Mirsky, A. Shabtai, B. Shapira, Y. Elovici, and L. Rokach. "Anomaly Detection for Smartphone Data Streams". In: *Pervasive and Mobile Computing* 35, 2017, pp. 83–107.

[135] A. H. Mirza and S. Cosan. "Computer Network Intrusion Detection using Sequential LSTM Neural Networks Autoencoders". In: *2018 26th signal processing and communications applications conference (SIU)*. IEEE. 2018, pp. 1–4.

[136] A. Mitseva, J. Pennekamp, J. Lohmöller, T. Ziemann, C. Hoerchner, K. Wehrle, and A. Panchenko. "POSTER: How Dangerous is My Click? Boosting Website Fingerprinting By Considering Sequences of Webpages". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 2411–2413.

[137] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica. "Ray: A Distributed Framework for Emerging AI Applications". In: *USENIX Symposium on Operating Systems Design and Implementation*. OSDI'18. 2018, pp. 561–577.

[138] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. "Network Intrusion Detection". In: *IEEE network* 8.3, 1994, pp. 26–41.

[139] S. J. Murdoch and G. Danezis. "Low-Cost Traffic Analysis of Tor". In: *IEEE Symposium on Security and Privacy*. S&P '05. IEEE, May 2005, pp. 183–195.

[140] K. Musgrave, S. Belongie, and S.-N. Lim. “A Metric Learning Reality Check”. In: *European Conference on Computer Vision*. Springer. 2020, pp. 681–699.

[141] C. Nadeau and Y. Bengio. “Inference for the Generalization Error”. In: *Machine learning* 52.3, 2003, pp. 239–281.

[142] A. Nadeem, V. Rimmer, W. Joosen, and S. Verwer. “Intelligent Malware Defenses”. In: *Security and Artificial Intelligence*. Springer, Apr. 2022, pp. 217–253.

[143] V. Nagarajan, A. Andreassen, and B. Neyshabur. “Understanding the Failure Modes of Out-Of-Distribution Generalization”. In: *International Conference on Learning Representations*. 2021.

[144] V. Nair and G. E. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *Proceedings of the 27th International Conference on Machine Learning (ICML)*. Ed. by J. Fürnkranz and T. Joachims. Omnipress, 2010, pp. 807–814. URL: `http://www.icml2010.org/papers/432.pdf`.

[145] M. Nasr, A. Bahramali, and A. Houmansadr. “DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning”. In: *ACM Conference on Computer and Communications Security*. CCS ’18. ACM, 2018, pp. 1962–1976.

[146] M. Nasr, A. Houmansadr, and A. Mazumdar. “Compressive Traffic Analysis: A New Paradigm for Scalable Traffic Analysis”. In: *ACM Conference on Computer and Communications Security*. CCS ’17. ACM, Oct. 2017, pp. 2053–2069.

[147] J. O. Nehinbe. “A Critical Evaluation of Datasets for Investigating IDSs and IPSs Researches”. In: *2011 IEEE 10th International Conference on Cybernetic Intelligent Systems (CIS)*. IEEE. 2011, pp. 92–97.

[148] H. T. Nguyen, K. Franke, and S. Petrovic. “Feature Extraction Methods for Intrusion Detection Systems”. In: *Threats, Countermeasures, and Advances in Applied Information Security*. IGI Global, 2012, pp. 23–52.

[149] Q. P. Nguyen, K. W. Lim, D. M. Divakaran, K. H. Low, and M. C. Chan. “GEE: A Gradient-Based Explainable Variational Autoencoder for Network Anomaly Detection”. In: *2019 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2019, pp. 91–99.

[150] J. Ning, G. S. Poh, J.-C. Loh, J. Chia, and E.-C. Chang. “PrivDPI: Privacy-Preserving Encrypted Traffic Inspection with Reusable Obfuscated Rules”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 1657–1670.

[151] S. E. Oh, N. Mathews, M. S. Rahman, M. Wright, and N. Hopper. "GANDaLF: GAN for Data-Limited Fingerprinting". In: *Privacy Enhancing Technologies Symposium* 2, 2021, pp. 305–322.

[152] S. E. Oh, S. Sunkam, and N. Hopper. "p1-fp: Extraction, Classification, and Prediction of Website Fingerprints with Deep Learning". In: *Proceedings on Privacy Enhancing Technologies* 2019.3, 2019, pp. 191–209.

[153] P. C. van Oorschot. "Intrusion Detection and Network-Based Attacks". In: *Computer Security and the Internet*. Springer, 2020, pp. 309–338.

[154] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Pennekamp, K. Wehrle, and T. Engel. "Website Fingerprinting at Internet Scale". In: *Network and Distributed System Security Symposium*. NDSS '16. The Internet Society, Feb. 2018.

[155] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. "Website Fingerprinting in Onion Routing Based Anonymization Networks". In: *ACM Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2011, pp. 103–114.

[156] M. Panda, A. Abraham, and M. R. Patra. "Discriminative Multinomial Naive Bayes for Network Intrusion Detection". In: *2010 Sixth International Conference on Information Assurance and Security*. IEEE. 2010, pp. 5–10.

[157] C. Pascoal, M. R. De Oliveira, R. Valadas, P. Filzmoser, P. Salvador, and A. Pacheco. "Robust Feature Selection and Robust PCA for Internet Traffic Anomaly Detection". In: *2012 Proceedings IEEE Infocom*. IEEE. 2012, pp. 1755–1763.

[158] V. Paxson. "Bro: A System for Detecting Network Intruders in Real-Time". In: *Computer networks* 31.23-24, 1999, pp. 2435–2463.

[159] D. Preuveneers, V. Rimmer, I. Tsingenopoulos, J. Spooren, W. Joosen, and E. Ilie-Zudor. "Chained Anomaly Detection Models for Federated Learning: An Intrusion Detection Case Study". In: *Applied Sciences* 8.12, Dec. 2018, pp. 1–21.

[160] T. T. Project. *Ethical Tor Research: Guidelines*. Jan. 2019. URL: `https://blog.torproject.org/blog/ethical-tor-research-guidelines`.

[161] T. T. Project. *The Onion Router*. Feb. 2021. URL: `https://www.torproject.org`.

[162] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence. *Dataset Shift in Machine Learning*. The MIT Press, 2009.

[163] M. S. Rahman, P. Sirinam, N. Mathews, K. G. Gangadhara, and M. Wright. "Tik-Tok: The utility of packet timing in website fingerprinting attacks". In: *Proceedings on Privacy Enhancing Technologies* 2020.3, 2020, pp. 5–24.

[164] A. Ramamoorthi, T. Subbulakshmi, and S. M. Shalinie. "Real Time Detection and Classification of DDoS Attacks using Enhanced SVM with String Kernels". In: *2011 International Conference on Recent Trends in Information Technology (ICRTIT)*. IEEE. 2011, pp. 91–96.

[165] M. A. Rassam, M. A. Maarof, and A. Zainal. "Adaptive and Online Data Anomaly Detection for Wireless Sensor Systems". In: *Knowledge-Based Systems* 60, 2014, pp. 44–57.

[166] H. Raza, G. Prasad, and Y. Li. "EWMA Model Based Shift-Detection Methods for Detecting Covariate Shifts in Non-Stationary Environments". In: *Pattern Recognition* 48.3, 2015, pp. 659–669.

[167] M. T. Ribeiro, S. Singh, and C. Guestrin. "Model-Agnostic Interpretability of Machine Learning". In: *arXiv preprint arXiv:1606.05386*, 2016.

[168] V. Rimmer. "Deep Learning Website Fingerprinting". MA thesis. KU Leuven, 2016.

[169] V. Rimmer, A. Nadeem, S. Verwer, D. Preuveneers, and W. Joosen. "Open-World Network Intrusion Detection". In: *Security and Artificial Intelligence*. Springer, Apr. 2022, pp. 254–283.

[170] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen. "Automated Website Fingerprinting through Deep Learning". In: *Network and Distributed System Security Symposium*. NDSS '18. The Internet Society, Feb. 2018, pp. 1–15.

[171] V. Rimmer, T. Schnitzler, T. Van Goethem, A. R. Romero, W. Joosen, and K. Kohls. "Trace Oddity: Methodologies for Data-Driven Traffic Analysis on Tor". In: *Proceedings on Privacy Enhancing Technologies* 3, July 2022, pp. 314–335.

[172] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho. "A Survey of Network-Based Intrusion Detection Data Sets". In: *Computers & Security* 86, 2019, pp. 147–167.

[173] M. Roesch et al. "Snort: Lightweight Intrusion Detection for Networks." In: *Lisa*. Vol. 99. 1. 1999, pp. 229–238.

[174] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft. "Deep One-Class Classification". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4393–4402.

[175] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning Representations by Back-Propagating Errors". In: *nature* 323.6088, 1986, pp. 533–536.

[176] S. Sagawa, A. Raghunathan, P. W. Koh, and P. Liang. "An Investigation of Why Overparameterization Exacerbates Spurious Correlations". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 8346–8356.

[177] M. N. Sakib and C.-T. Huang. "Using Anomaly Detection Based Techniques to Detect HTTP-Based Botnet C&C Traffic". In: *2016 IEEE International Conference on Communications (ICC)*. IEEE. 2016, pp. 1–6.

[178] M. Sayrafi. *Introducing the Cloudflare Onion Service*. Sept. 2019. URL: `https://blog.cloudflare.com/cloudflare-onion-service/`.

[179] P. Schneider and K. Böttinger. "High-Performance Unsupervised Anomaly Detection for Cyber-Physical System Networks". In: *Proceedings of the 2018 workshop on cyber-physical systems security and privacy*. 2018, pp. 1–12.

[180] T. Schnitzler, C. Pöpper, M. Dürmuth, and K. Kohls. "We Built This Circuit: Exploring Threat Vectors in Circuit Establishment in Tor". In: *IEEE European Symposium on Security and Privacy*. EuroS&P '21. IEEE, 2021.

[181] F. Schroff, D. Kalenichenko, and J. Philbin. "Facenet: A Unified Embedding for Face Recognition and Clustering". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.

[182] R. Schuster, V. Shmatikov, and E. Tromer. "Beauty and the Burst: Remote Identification of Encrypted Video Streams". In: *USENIX Security Symposium*. 2017, pp. 1357–1374.

[183] A. Shafiei, V. Rimmer, I. Tsingenopoulos, L. Desmet, and W. Joosen. "Position Paper: On Advancing Adversarial Malware Generation Using Dynamic Features". In: *Proceedings of the 1st Workshop on Robust Malware Analysis*. May 2022, pp. 15–20.

[184] S. Shan, A. N. Bhagoji, H. Zheng, and B. Y. Zhao. "Patch-Based Defenses Against Web Fingerprinting Attacks". In: *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*. 2021, pp. 97–109.

[185] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani. "Towards a Reliable Intrusion Detection Benchmark Dataset". In: *Software Networking* 2018.1, 2018, pp. 177–200.

[186] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: *ICISSP*. 2018, pp. 108–116.

[187] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. "Blindbox: Deep Packet Inspection over Encrypted Traffic". In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 2015, pp. 213–226.

[188] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani. "Toward Developing a Systematic Approach to Generate Benchmark Datasets for Intrusion Detection". In: *computers & security* 31.3, 2012, pp. 357–374.

[189] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi. "A Deep Learning Approach to Network Intrusion Detection". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 2.1, 2018, pp. 41–50.

[190] A. Shrikumar, P. Greenside, and A. Kundaje. "Learning Important Features Through Propagating Activation Differences". In: *International conference on machine learning*. PMLR. 2017, pp. 3145–3153.

[191] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. "The Hadoop Distributed File System". In: *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. IEEE. 2010, pp. 1–10.

[192] S. Singla, B. Nushi, S. Shah, E. Kamar, and E. Horvitz. "Understanding Failures of Deep Networks via Robust Feature Extraction". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 12853–12862.

[193] P. Sirinam, M. Imani, M. Juarez, and M. Wright. "Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning". In: *ACM Conference on Computer and Communications Security*. CCS '18. ACM, 2018, pp. 1928–1943.

[194] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright. "Triplet Fingerprinting: More Practical and Portable Website Fingerprinting with N-Shot Learning". In: *ACM Conference on Computer and Communications Security*. CCS '19. ACM, 2019, pp. 1131–1148.

[195] J. Snoek, H. Larochelle, and R. P. Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in neural information processing systems* 25, 2012.

[196] R. Sommer and V. Paxson. "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection". In: *2010 IEEE symposium on security and privacy*. IEEE. 2010, pp. 305–316.

[197] A. Sperotto, M. Mandjes, R. Sadre, P.-T. de Boer, and A. Pras. "Autonomic Parameter Tuning of Anomaly-Based IDSs: An SSH Case Study". In: *IEEE Transactions on Network and Service Management* 9.2, 2012, pp. 128–141.

[198] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. "An Overview of IP Flow-Based Intrusion Detection". In: *IEEE communications surveys & tutorials* 12.3, 2010, pp. 343–356.

[199] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.1, 2014, pp. 1929–1958.

[200] M.-Y. Su. "Real-Time Anomaly Detection Systems for Denial-of-Service Attacks by Weighted k-Nearest-Neighbor Classifiers". In: *Expert Systems with Applications* 38.4, 2011, pp. 3492–3498.

[201] K. Sun and F. Nielsen. "Lightlike Neuromanifolds, Occam's Razor and Deep Learning". In: *arXiv preprint arXiv:1905.11027*, 2019.

[202] Q. Sun, D. R. Simon, and Y. M. Wang. "Statistical Identification of Encrypted Web Browsing Traffic". In: *IEEE Symposium on Security and Privacy*. S&P '02. IEEE, 2002, pp. 19–30.

[203] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal. "RAPTOR: Routing attacks on privacy in Tor". In: *USENIX Security Symposium*. USENIX Association, 2015, pp. 271–286.

[204] M. Sundararajan, A. Taly, and Q. Yan. "Axiomatic Attribution for Deep Networks". In: *International conference on machine learning*. PMLR. 2017, pp. 3319–3328.

[205] *Suricata IDS [Online]*. Accessed: Jun. 1, 2020. 2010. URL: `https://suricata-ids.org/`.

[206] S. Suthaharan. "Big Data Classification: Problems and Challenges in Network Intrusion Prediction with Machine Learning". In: *ACM SIGMETRICS Performance Evaluation Review* 41.4, 2014, pp. 70–73.

[207] M. Tavallaee, N. Stakhanova, and A. A. Ghorbani. "Toward Credible Evaluation of Anomaly-Based Intrusion-Detection Methods". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40.5, 2010, pp. 516–524.

[208] T. D. Team. "Theano: A Python Framework for Fast Computation of Mathematical Expressions". In: *arXiv e-prints* abs/1605.02688, May 2016.

[209] C. Thomas. “Improving Intrusion Detection for Imbalanced Network Traffic”. In: *Security and Communication Networks* 6.3, 2013, pp. 309–324.

[210] G. Valle-Perez, C. Q. Camargo, and A. A. Louis. “Deep Learning Generalizes Because the Parameter-Function Map is Biased Towards Simple Functions”. In: *arXiv preprint arXiv:1805.08522*, 2018.

[211] T. Van hamme, V. Rimmer, D. Preuveneers, W. Joosen, M. Mustafa, A. Abidin, and E. Argones Rua. “Frictionless Authentication Systems: Emerging Trends, Research Challenges and Opportunities”. In: *11th International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2017)*. IARIA, Sept. 2017, pp. 1–5.

[212] G. K. Varadarajan and M. Santander Peláez. “Web Application Attack Analysis using Bro IDS”. In: *SANS Institute* 90, 2012.

[213] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer. “Taxonomy and Survey of Collaborative Intrusion Detection”. In: *ACM Computing Surveys (CSUR)* 47.4, 2015, pp. 1–33.

[214] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. “Attention is All You Need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[215] A. Veicht, C. Renggli, and D. Barradas. “DeepSE-WF: Unified Security Estimation for Website Fingerprinting Defenses”. In: *arXiv preprint arXiv:2203.04428*, 2022.

[216] P. Velarde-Alvarado, C. Vargas-Rosales, R. Martinez-Pelaez, H. Toral-Cruz, and A. F. Martinez-Herrera. “An Unsupervised Approach for Traffic Trace Sanitization based on the Entropy Spaces”. In: *Telecommunication Systems* 61.3, 2016, pp. 609–626.

[217] E. K. Viegas, A. O. Santin, and L. S. Oliveira. “Toward a Reliable Anomaly-Based Intrusion Detection in Real-World Environments”. In: *Computer Networks* 127, 2017, pp. 200–216.

[218] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. “Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion”. In: *Journal of Machine Learning Research* 11.Dec, 2010, pp. 3371–3408.

[219] K. Wang, J. J. Parekh, and S. J. Stolfo. “Anagram: A Content Anomaly Detector Resistant to Mimicry Attack”. In: *International workshop on recent advances in intrusion detection*. Springer. 2006, pp. 226–248.

[220] K. Wang and S. J. Stolfo. “Anomalous Payload-Based Network Intrusion Detection”. In: *International workshop on recent advances in intrusion detection*. Springer. 2004, pp. 203–222.

[221] M. Wang, Y. Li, X. Wang, T. Liu, J. Shi, and M. Chen. "2ch-TCN: A Website Fingerprinting Attack over Tor using 2-Channel Temporal Convolutional Networks". In: *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2020, pp. 1–7.

[222] T. Wang. "High Precision Open-World Website Fingerprinting". In: *IEEE Symposium on Security and Privacy*. S&P '20. IEEE. The Internet Society, 2020, pp. 152–167.

[223] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. "Effective Attacks and Provable Defenses for Website Fingerprinting". In: *USENIX Security Symposium*. USENIX '14. USENIX Association, Aug. 2014, pp. 271–286.

[224] T. Wang and I. Goldberg. "Improved Website Fingerprinting on Tor". In: *ACM Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2013, pp. 201–212.

[225] T. Wang and I. Goldberg. "On Realistically Attacking Tor with Website Fingerprinting". In: *Proceedings on Privacy Enhancing Technologies (PoPETs)*. De Gruyter Open, 2016, pp. 21–36.

[226] T. Wang and I. Goldberg. "Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks". In: *USENIX Security Symposium*. USENIX Association, 2017, pp. 1375–1390. ISBN: 978-1-931971-40-9.

[227] W. Wang, T. Guyet, R. Quiniou, M.-O. Cordier, F. Masseglia, and X. Zhang. "Autonomic Intrusion Detection: Adaptively Detecting Anomalies over Unlabeled Audit Data Streams in Computer Networks". In: *Knowledge-Based Systems* 70, 2014, pp. 103–117.

[228] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang. "End-to-End Encrypted Traffic Classification with One-Dimensional Convolution Neural Networks". In: *2017 IEEE international conference on intelligence and security informatics (ISI)*. IEEE. 2017, pp. 43–48.

[229] Z. Wang. "The Applications of Deep Learning on Traffic Identification". In: *BlackHat USA*, 2015.

[230] A. Warnecke, D. Arp, C. Wressnegger, and K. Rieck. "Evaluating Explanation Methods for Deep Learning in Security". In: *IEEE European Symposium on Security and Privacy*. EuroS&P '20. IEEE. 2020, pp. 158–174.

[231] L. Wu, Z. Zhu, et al. "Towards Understanding Generalization of Deep Learning: Perspective of Loss Landscapes". In: *arXiv preprint arXiv:1706.10239*, 2017.

[232] K. Xu, Z.-L. Zhang, and S. Bhattacharyya. "Profiling Internet Backbone Traffic: Behavior Models and Applications". In: *ACM SIGCOMM Computer Communication Review* 35.4, 2005, pp. 169–180.

[233] Y. Xu, T. Wang, Q. Li, Q. Gong, Y. Chen, and Y. Jiang. "A Multi-Tab Website Fingerprinting Attack". In: *Proceedings of the 34th Annual Computer Security Applications Conference*. 2018, pp. 327–341.

[234] Y. Yu, J. Long, and Z. Cai. "Session-Based Network Intrusion Detection using a Deep Learning Architecture". In: *International conference on modeling decisions for artificial intelligence*. Springer. 2017, pp. 144–155.

[235] W. T. Yue and M. Çakanyıldırım. "A Cost-Based Analysis of Intrusion Detection System Configuration under Active or Passive Response". In: *Decision Support Systems* 50.1, 2010, pp. 21–31.

[236] B. B. Zarpelão, R. S. Miani, C. T. Kawakani, and S. C. de Alvarenga. "A Survey of Intrusion Detection in Internet of Things". In: *Journal of Network and Computer Applications* 84, 2017, pp. 25–37.

[237] *Zeek IDS [Online]*. Accessed: Jul. 12, 2020. 2018. URL: `https://zeek.org/`.

[238] M. D. Zeiler and R. Fergus. "Visualizing and Understanding Convolutional Networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[239] J. Zhang, H. Li, Q. Gao, H. Wang, and Y. Luo. "Detecting Anomalies from Big Network Traffic Data using an Adaptive Detection Approach". In: *Information Sciences* 318, 2015, pp. 91–110.

[240] J. Zhang, M. Zulkernine, and A. Haque. "Random-Forests-based Network Intrusion Detection Systems". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.5, 2008, pp. 649–659.

[241] Y. Zhang, X. Chen, L. Jin, X. Wang, and D. Guo. "Network Intrusion Detection: Based on Deep Hierarchical Network and Original Flow Data". In: *IEEE Access* 7, 2019, pp. 37004–37016.

List of Publications

- V. Rimmer, T. Schnitzler, T. Van Goethem, A. R. Romero, W. Joosen, and K. Kohls. “Trace Oddity: Methodologies for Data-Driven Traffic Analysis on Tor”. In: *Proceedings on Privacy Enhancing Technologies* 3, July 2022, pp. 314–335.
- A. Shafiei, V. Rimmer, I. Tsingenopoulos, L. Desmet, and W. Joosen. “Position Paper: On Advancing Adversarial Malware Generation Using Dynamic Features”. In: *Proceedings of the 1st Workshop on Robust Malware Analysis*. May 2022, pp. 15–20.
- A. Nadeem, V. Rimmer, W. Joosen, and S. Verwer. “Intelligent Malware Defenses”. In: *Security and Artificial Intelligence*. Springer, Apr. 2022, pp. 217–253.
- V. Rimmer, A. Nadeem, S. Verwer, D. Preuveneers, and W. Joosen. “Open-World Network Intrusion Detection”. In: *Security and Artificial Intelligence*. Springer, Apr. 2022, pp. 254–283.
- G. Engelen, V. Rimmer, and W. Joosen. “Troubleshooting an Intrusion Detection Dataset: The CICIDS2017 Case Study”. In: *2021 IEEE Security and Privacy Workshops (SPW)*. May 2021, pp. 7–12.
- D. Preuveneers, V. Rimmer, I. Tsingenopoulos, J. Spooren, W. Joosen, and E. Ilie-Zudor. “Chained Anomaly Detection Models for Federated Learning: An Intrusion Detection Case Study”. In: *Applied Sciences* 8.12, Dec. 2018, pp. 1–21.
- G. Garofalo, V. Rimmer, T. Van hamme, D. Preuveneers, W. Joosen, et al. “Fishy Faces: Crafting Adversarial Images to Poison Face Authentication”. In: *USENIX Workshop on Offensive Technologies*. WOOT ’18. Aug. 2018, pp. 1–12.

- V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen. "Automated Website Fingerprinting through Deep Learning". In: *Network and Distributed System Security Symposium*. NDSS '18. The Internet Society, Feb. 2018, pp. 1–15.
- D. Preuveneers, A. Ramakrishnan, T. Van hamme, V. Rimmer, Y. Berbers, and W. Joosen. "A Survey on Applying Machine Learning Techniques for Behavioral Awareness". In: *State of the Art in AI Applied to Ambient Intelligence*. IOS Press, Oct. 2017, pp. 1–34.
- T. Van hamme, V. Rimmer, D. Preuveneers, W. Joosen, M. Mustafa, A. Abidin, and E. Argones Rua. "Frictionless Authentication Systems: Emerging Trends, Research Challenges and Opportunities". In: *11th International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2017)*. IARIA, Sept. 2017, pp. 1–5.



FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
IMEC-DISTRINET
Celestijnenlaan 200A box 2402
B-3001 Leuven
vera.rimmer@kuleuven.be
http://www.distrinet.cs.kuleuven.be