Towards the Future Internet
G. Tselentis et al. (Eds.)
IOS Press, 2009
© 2009 The authors and IOS Press. All rights reserved.
doi:10.3233/978-1-60750-007-0-35

A Security Architecture for Web 2.0 Applications

Lieven Desmet[1], Wouter Joosen[1], Fabio Massacci[2], Katsiaryna Naliuka[2], Pieter Philippaerts[1], Frank Piessens[1], Ida Siahaan[2], Dries Vanoverberghe[1]

[1] DistriNet Research Group, Department of Computer Science
Katholieke Universiteit Leuven, Celestijnlaan 200A, B-3001 Leuven, Belgium
[2] Department of Information and Communication Technology
Universit di Trento, Via Sommarive 14, I-38050 Povo (Trento), Italy

Abstract. The problem of supporting the secure execution of potentially malicious third-party applications has received a considerable amount of attention in the past decade. In this paper we describe a security architecture for Web 2.0 applications that supports the flexible integration of a variety of advanced technologies for such secure execution of applications, including run-time monitoring, static verification and proof-carrying code. The architecture also supports the execution of legacy applications that have not been developed to take advantage of our architecture, though it can provide better performance and additional services for applications that are architecture-aware. A prototype of the proposed architecture has been built that offers substantial security benefits compared to standard (state-of-practice) security architectures, even for legacy applications.

1 Introduction

The business model of traditional web-services architectures is based on a very simple assumption: the good guys develop their .NET or Java application, expose it on the web (normally to make money as an application service provider), and then spend the rest of their life letting other good guys use it while stopping bad guys from misusing it.

The business trend of outsourcing the non-core part of business processes [8] or the construction of virtual organizations [10] have slightly complicated this initially simple picture. With virtual organizations for fulfilling the same high level goal

1. different service components are dynamically chosen (possibly using different data) and
2. different partners are chosen (possibly with different service level agreements or trust levels).

This requires different security models, policies, infrastructures and trust establishment mechanisms (see e.g. [13, 12]). A large part of the WS security standards (WS-Federation, WS-Trust, WS-Security) are geared to solve some of these problems.

Still, the assumption is the same: *the application developer and the platform owner are on the same side of trust border*. Traditional books on secure coding [11] or the .NET security handbook [14] are permeated with this assumption. However, in the landscape of Web 2.0, users are downloading a multitude of communicating applications ranging from P2P clients to desktop search engines, each plowing through the

user's platform, and communicating with the rest of the world. Most of these applications have been developed by people that the user never knew existed, before installing the application.

The (once) enthusiastic customers of UK Channel 4 on demand services 4oD might tell how use of the third-party services affects the customer's experience [18]. Downloading a client that allows you to watch movies at a very cheap rate seems like an excellent deal. Only in the fine print of the legal terms of use (well hidden from the download now, nowhere in the FAQs and after a long scrolling down of legalese) you find something you would like to know:

> If you download Content to your computer, during the Licence Period, we may upload this from your computer (using part of your upstream bandwidth) for the purpose of transferring Content to other users of the Service. Please contact your Internet Service Provider ("ISP") if you have any queries on this.

As one of the unfortunate users of the system noticed, there is no need of contacting your ISP. He will contact you pretty soon.

To deal with the untrusted code, either .NET [14] or Java [15] exploit the mechanism of permissions. Permissions are assigned to enable execution of potentially dangerous or costly functionality, such as starting various types of connections. The drawback of permissions is that after assigning a permission the user has very limited control over how the permission is used. The consequence is that either applications are sandboxed (and thus can do almost nothing), or the user decides that they are trusted and once the permission is received then they can do almost everything.

The mechanism of signed assemblies from trusted third parties might solve the problem. Unfortunately a signature means that the signatory vouches for the software, but there is no clear definition of what guarantees it offers. The 4oD example or the incidents in the mobile phone domain [21] show that this security model is inappropriate.

In this paper, we describe the architecture of the runtime environment on the Web 2.0 platform that we have already developed for .NET (both the desktop and the compact edition). The architecture integrates in a very flexible way several state-of-the-art policy enforcement technologies, such as proof-carrying code and inlined reference monitors. In addition, the security architecture offers additional support for application contracts and the security-by-contract paradigm. Thanks to the combination of different enforcement techniques and the support for application contracts, our security architecture is able to provide policy enforcement for legacy applications, as well as architecture-aware applications. However, the latter set of applications have a smaller runtime performance penalty, which is an important characteristic for resource-restricted environments such as mobile Web 2.0 platforms. In addition, a first prototype implementation of the proposed security architecture is available for Windows based desktops, and for Windows mobile platforms with the .NET Compact framework (so it is also suitable for mobile devices).

The remainder of the paper is structured as follows. Section 2 provides some background information on the security-by-contract paradigm, existing policy enforcement techniques, and policy languages. Next, our flexible security architecture for Web 2.0 platforms is presented in Section 3, and Section 4 describes our prototype implementation. In Section 5, the advantages and disadvantages of the presented architecture are discussed. Finally, the presented work is related to existing research, and we offer a conclusion.

2 Background

The architecture described in this paper is largely based on the research results of the European project S3MS [23]. In this section, we describe the key notion of *security-by-contract* underlying the S3MS project, and we briefly discuss the policy enforcement techniques and policy languages considered in that project.

A key ingredient in the S3MS approach is the notion of "security-by-contract". Web 2.0 applications can possibly come with a *security contract* that specifies their security-relevant behavior [4]. Technically, a contract is a security automaton in the sense of Schneider and Erlingsson [6], and it specifies an upper bound on the security-relevant behavior of the application: the sequences of security-relevant events that an application can generate are all in the language accepted by the security automaton. Web 2.0 platforms are equipped with a *security policy*, a security automaton that specifies the behavior that is considered acceptable by the platform owner. The key task of the S3MS environment is to ensure that all applications will comply with the platform security policy. To achieve this, the system can make use of the contract associated with the application (if it has one), and of a variety of policy enforcement technologies.

2.1 Policy Enforcement Techniques

The research community has developed a variety of countermeasures to address the threat of untrusted code. These countermeasures are typically based on runtime monitoring [6], static analysis [19], or a combination of both [28]. We briefly review here the technologies supported in the S3MS system. It must be noted, however, that the system so new technologies can be implemented as needed.

Cryptographic signatures. The simplest way to solve the lack of trust is to use *cryptographic signatures.* The application is signed, and is distributed along with this signature. After receiving this application, the signature can be used to verify the source and integrity of the application. Traditionally, when a third party signs an application, it means that this third party certifies the application is well-behaved. Adding the notion of a contract, as is done in the S3MS approach, allows us to add more meaning to claims on well-behavior: the signature means that the application respects the supplied contract.

Inline reference monitoring. With *inline reference monitoring* [6], a program is rewritten so that security checks are inserted inside an untrusted application. When the application is executed, these checks monitor the behavior of the application and prevent it from violating the policy. It is an easy way to secure an application when it has not been developed with a security policy in mind or when all other techniques have failed. The biggest challenge for inline reference monitors is to make sure that the application can not circumvent the inlined security checks.

Proof-carrying code. An alternative way to enforce a security policy is to statically verify that an application does not violate this policy. As producing the proof is normally too complicated to be done on the user side, the *proof carrying code* concept [19], allows the verification to be be done off-line by the developer, or by an expert in the field. Then the application is distributed together with the proof. Before allowing the execution of the application, a proof-checker verifies that the proof is correct for the

application. Because proof-checking is usually much easier than making the proof, this step becomes feasible on Web 2.0 platforms.

Contract-policy matching. Finally, when application contracts (called application models in [25]) are available, *contract-policy matching* [6, 17, 25] is an approach to decide whether or not the contract is acceptable. At the deployment of the application the contract acts as an intermediary between the application and the policy of the platform. First, a matching step checks whether all security-relevant behavior allowed by the contract is also allowed by the policy. If this is the case, any of the other enforcement techniques can be used to make sure that the application complies to the contract (as the contract of the application is known in advance this step can be made off-line).

2.2 Policy Languages

In this paper, we make a clear distinction between application contracts and platform policies. Both are security automata, but the first ones are associated with a particular application, while the latter ones are associated with a platform.

A security automaton [24] is a Büchi automaton – the extension of the notion of finite state automaton to infinite inputs. A security automaton specifies the set of acceptable sequences of *security relevant events* as the language accepted by the automaton.

In the S3MS system, a policy identifies a subset of the methods of the platform API as security relevant methods. Typical examples are the methods to open network connections or to read files. Security relevant events in the S3MS system are the invocations of these methods by the untrusted application, as well as the returns from such invocations. Hence, a security automaton specifies the acceptable sequences of method invocations and returns on security relevant methods from the platform API.

Security automata have to be specified by means of a policy language. The S3MS system is designed to support multiple policy languages, including policy languages that support multiple runs of the same application. The actual prototype implementation supports already two languages: automata-based specification language ConSpec [2] and logic-based language 2D-LTL [16].

3 System Architecture

In this section, our service-oriented security architecture for Web 2.0 platforms is presented. First, we enumerate the most important architectural requirements for the security architecture. Next, subsection 3.1 gives an overview of our architecture, and highlights three important architectural scenario's. The following three subsections discuss some architectural decisions in more detail.

Before presenting and discussing our flexible service-oriented security architecture, the most important architectural requirements are briefly discussed.

Secure execution of third-party applications The architecture should give high assurance that applications that have been processed by the security system can never break the platform security policy. This is the key functional requirement of our architecture.

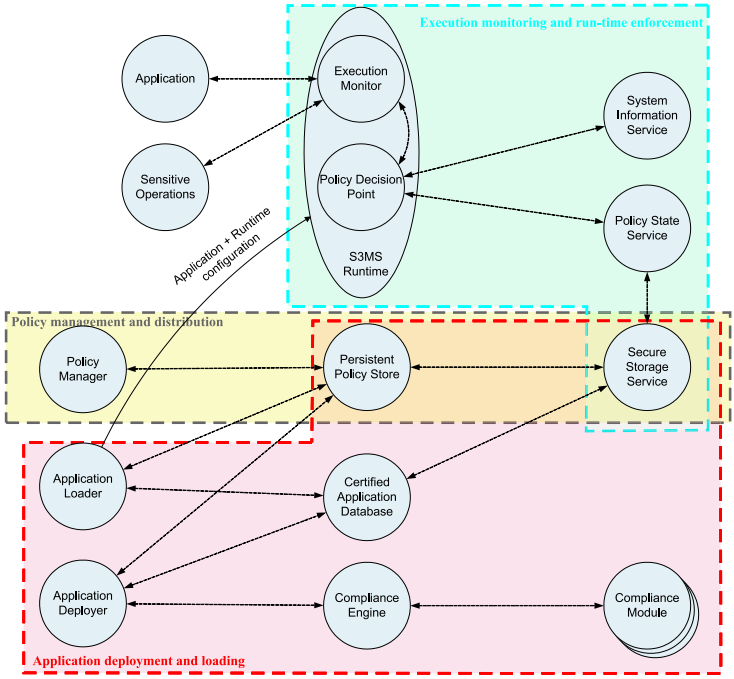

Fig. 1. Detailed architecture overview

Flexible integration of enforcement techniques The security architecture should integrate seamlessly the set of enforcement techniques discussed in Sec. 2. In addition, the security architecture should provide a flexible framework for adding, configuring or removing additional enforcement techniques.

Optimized for resource-restricted platforms The security architecture needs to be optimized for use on resource-restricted Web 2.0 platforms such as personal digital assistants or SmartPhones. These platform typically have limited memory and processing power, and restricted battery capacity. The architecture should secure the execution of applications with a minimal performance penalty during the application execution, without compromising security during network disconnectivity.

Compatible with legacy applications To be compatible with existing applications, it is important that the security architecture supports the secure execution of legacy applications that are unaware of the architecture. Of course, the fact that an application is architecture-unaware may impact performance.

In the following section, an overview of our security architecture for Web 2.0 platforms is presented. As will be explained further, each of the enumerated architectural requirements has impacted the overall architecture.

3.1 Overview

The security architecture is built upon the notion of "security-by-contract". Web 2.0 platforms can select a security policy from a list of available policies, specifying an upper bound on the security-relevant behavior of applications. In addition, applications can be distributed with a security contract, specifying their security-relevant behavior.

The three main scenarios are: policy management and distribution, application deployment and loading, and execution monitoring and runtime enforcement.

Policy management and distribution This scenario is responsible for the management of different platform policies, and their distribution to Web 2.0 platforms.

Application deployment and loading This scenario is responsible for verifying the compliance of a particular application with the platform policy before this application is executed.

Execution monitoring and runtime enforcement This scenario is responsible for enforcing the adherence of a running application to the policy of the platform in the case where the previous scenario has decided that this is necessary.

The three scenarios operate on two different platforms: on the platform of the policy provider and on the Web 2.0 platform.

Policy provider. Within the S3MS security architecture, the policies are managed by the *Policy Provider* and a specific policy can be pushed to a particular Web 2.0 platform. The policy provider could for instance be a company that supplies its employees with Web 2.0-capable devices, but wishes to enforce a uniform policy on all these devices. It could also be an advanced end-user that wants to manage the policy using a PC that can connect to his Web 2.0 platform.

Web 2.0 platform. The Web 2.0 platform stores the policy and is responsible for deploying, loading and running applications. If necessary, it also applies execution monitoring and run-time enforcement to ensure compliance to the device policy.

Figure 1 shows an architectural overview of the entire platform, and of the software entities that are involved in these three scenarios.

3.2 Policy Representations

Our architectural requirements ask for flexibility in policy enforcement techniques used, as well as for resource conscious implementations. However, the different enforcement techniques impose different constraints on optimized policy representations, and hence it is hard, or even impossible, to find one optimal representation that is suitable for each technique.

For instance, in a runtime monitor, the decision whether an event is allowed or aborted relies only on the current state of the policy. Therefore, an efficient representation for runtime enforcement only contains the current state, and methods for each event that check against the state, and update it. On the other hand, contract/policy matching checks whether or not the behavior allowed by the contract is a subset of the behavior allowed by the policy. For this task, a full graph representation may be required.

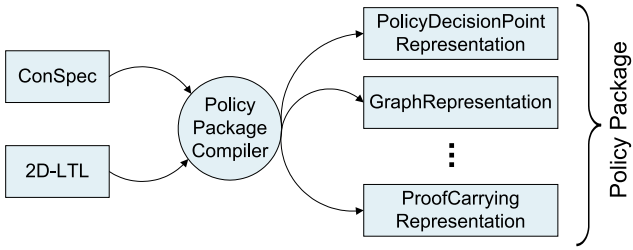

Fig. 2. Compilation of a policy package

To deal with this problem, our architecture introduces the notion of *policy packages*. A policy package is a set of optimized representations (each geared towards a different enforcement technique) of the same policy.

The policy writer is responsible for distributing the policy packages to the policy broker service. To do so, he uses a policy compiler to transform a given policy specification (e.g. in ConSpec or 2D-LTL) to a policy package (figure 2). This policy package is then sent to the policy broker. In a similar vein, representation of application contracts can be optimized towards different application-policy matching algorithms, and hence contracts are supplied in our architecture in the form of a similar contract package.

3.3 The Deployment Framework: Support for a Variety of Policy Enforcement Techniques.

In order to achieve a powerful security architecture that can incorporate a variety of state-of-the-art policy enforcement techniques, our architecture includes a very configurable and extensible compliance engine. At deployment time of an application, this compliance engine attempts to ensure the compliance of the application by means of all supported enforcement technologies. Each enforcement technology is represented as a *compliance module*.

Each compliance module selects the policy representation it will use and sends it, together with the application and optional metadata, to a certification technology service.

Each compliance verification technology is encapsulated in a *ComplianceModule*. To verify the compliance of an application with a policy, the *Process(PolicyPackage policy, Application app)* method is executed on such a compliance module. The module selects the policy representation it will use and sends it, together with the application and optional metadata, to a certification technology service. The result of the method indicates whether or not the compliance verification is successful. As a side effect of executing the process method, the application can be altered (e.g. instrumented with an inline reference monitor). The compliance engine instantiates the different compliance modules and applies them sequentially until the compliance of the application with the policy is ensured.

The order in which the compliance modules are applied, and their particular configuration is made policy-specific. This policy-specific configuration is part of the policy

package. In this way, policies can optimize the certification process by favoring or excluding compliance modules (e.g. because they are optimal for the given policy, or because they are inappropriate).

4 Prototype Implementation

We have implemented a first prototype of this security architecture in the full .NET Framework, and in the .NET Compact Framework on Windows Mobile 5. We've chosen to do an implementation on a mobile device, because handheld devices are becoming ever more popular and have already made an entrance to the Web 2.0 world (sometimes called *the Mobile Web 2.0*). In addition, using mobile devices forces us to consider resource-restricted platforms.

Our prototype includes policy compilers for both ConSpec, a policy specification language based on security automata, as well as 2D-LTL, a bi-dimensional temporal logic language. The compiler outputs a policy representation package that includes three policy representations, one suitable for inlining, one suitable for signature verification, and one suitable for matching. The corresponding compliance modules that use these representations are also implemented. Compliance modules need not be aware of the source policy language. For instance, our runtime execution monitor uses the same Policy Decision Point interface irrespectively of the policy specification language used.

Our current inliner implementation uses caller side inlining. Because caller side inlining needs to find the target of a method call statically, it is harder to ensure complete mediation. Under certain assumptions about the platform libraries, and with certain restrictions on the applications being monitored, our inlining algorithms has been proven correct [27]. The main assumption about the platform library is that it will not create delegates (function pointers) to security relevant methods and return these to the untrusted application. This seems to be a realistic assumption for the .NET libraries. The main restriction we impose on applications is that we forbid the use of reflection, a common restriction in code access security systems.

A final noteworthy implementation aspect of our mobile prototype is the way we ensure that only compliant applications can be executed on the mobile device, i.e. only after the application successfully passes the deployment scenario. Instead of maintaining and enforcing a Certified Application Database, we decided to rely on the underlying security model of Windows Mobile 5.0 in our prototype. The *Locked or Third-Party-Signed* configuration in Windows Mobile allows a mobile device to be locked so that only applications signed with a trusted certificate can run [1]. By adding a policy-specific certificate to the trusted key store, and by signing applications with that certificate after successfully passing the deployment scenario, we ensure that non-compliant applications will never be executed on the protected mobile device.

5 Discussion

In this section, we offer a brief preliminary evaluation of the presented security architecture based on the architectural requirements set forth in Section 3.

Secure execution of third-party applications Our architecture assumes that the individual compliance modules and certification technology services are secure: a buggy or unreliable implementation can validate an application that does not comply with the platform policy. This is a weakness, but the cost of building in redundancy (e.g. requiring two independent compliance modules to validate an application) is too high. Apart from this weakness, our architecture supports high assurance of security through a simple and well-defined compliance validation process, and through the precise definitions of the guarantees offered by security checks. An example is the treatment of cryptographic signatures. Our security architecture relies on cryptographic signatures in several places. But a key difference with the use of cryptographic signatures in the current .NET and Java security architectures is the fact that the semantics of a signature in our system are always clearly and unambiguously defined. A signature on an application with a contract means that the trusted third party attests to the fact that the application complies with the contract, and this is a formally defined statement.

Flexible integration of enforcement techniques The architecture incorporates different techniques such as cryptographic signatures, contract/policy matching and inlining of a reference monitor. Thanks to the pluggable compliance modules and the concept of policy packages the framework can easily be extended with additional enforcement technologies. In addition, the policy configuration allows for policy-specific configuration of the different compliance modules, including configuration of the order in which they are applied to applications.

Optimized for resource-restricted platforms Proving that an application will never violate a give system policy is typically a relatively hard problem. This might become problematic because of the resource-restricted nature of Web 2.0 platforms. Thanks to the service-oriented architecture, most of this work can be outsourced from the platform to the contract certifier service. These high-performance certifier servers can process multiple requests simultaneous and can sometimes, depending on the policy enforcement technique that's being used, use caching mechanisms to speed up the certification of an application by using results obtained from the certification of a previous application.

Compatible with legacy applications Because of the use of a general-applicable fallback compliance module (e.g. inlining of a reference monitor), the architecture can also ensure security for architecture-unaware legacy applications.

Based on this preliminary evaluation, the presented architecture looks promising. However, a more in-depth architectural evaluation and validation is necessary for a more grounded conclusion. We see three important tracks for further evaluation of the presented architecture.

First, *an extensive architectural evaluation* of the proposed architecture is necessary. For instance, an architectural trade-off analysis (such as ATAM [22]) with the different stakeholders involved (such as end users, telecom operators, Web 2.0 application developers and vendors, platform vendors and security experts), can evaluate and refine several architectural trade-offs. Second, it is necessary to perform an *end-to-end threat analysis* of the proposed architecture. Based on these results, a risk assessment will identify the most important security risks and will provide additional input for the

architectural trade-off analysis. Third, a *further integration of existing enforcement techniques* in the prototype architecture is needed to validate the flexibility of the framework design.

6 Related Work

There is a huge body of related work that deals with specific policy enforcement technologies for untrusted applications. This research area is too broad to discuss here. Some of the key technologies were briefly discussed in Section 2. A more complete survey of relevant technologies can be found in one of the deliverables of the S3MS project [26].

Even more closely related are those research projects that have designed and implemented working systems building on one or more of the technologies discussed above. Naccio [7] and PoET/PSlang [5] were pioneering implementations of runtime monitors. Polymer [3] is also based mainly on runtime monitoring, but the policy that is enforced can depend on the signatures that are present on the code. Model-carrying code (MCC) [25] is an enforcement technique that is very related to the contract matching based enforcement used in the S3MS project. In MCC, an applications comes with a *model* of its security relevant behavior, and hence models are basically the same as contracts. The MCC paper describes a system design where models are extracted from the application by the code producer. The code consumer uses the model to select a matching policy, and enforces the model at runtime. Mobile [9] is an extension to the .NET Common Intermediate Language that supports certified inline reference monitoring. Certifying compilers [20] use similar techniques like proof carrying code, but they include type system information instead of proofs.

We should also mention here the existing framework for enforcing information flow control in the Web 2.0 setting DIFSA-J [29]. This approach allows rewriting the third-party programs at the bytecode level to insert the inline reference monitor that controls access to the sensitive information in the databases and prevents leakage to undesirable parties.

7 Conclusion

We proposed a flexible security architecture for Web 2.0 platforms built upon the notion of "security-by-contract". In a very extensible way, the architecture integrates a variety of state-of-the art technologies for secure execution of Web 2.0 applications, and supports different policy specification languages. In addition, the proposed architecture also supports the secure execution of legacy applications, although a better runtime performance is achieved for security-by-contract-aware applications.

8 Acknowledgments

This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the European Union under the FP6 Programme.

References

1. Windows Mobile 5.0 application security. Available at `http://msdn2.microsoft.com/en-us/library/ms839681.aspx`, 2005.
2. I. Aktug and K. Naliuka. Conspec – a formal language for policy specification. In *Proc. of the 1st Int. Workshop on Run Time Enforcement for Mobile and Distributed Systems (REM2007)*, 2007.
3. Lujo Bauer, Jay Ligatti, and David Walker. Composing security policies with Polymer. In *Proc. of the ACM SIGPLAN 2005 Conf. on Prog. Lang. Design and Implementation*, pages 305–314, 2005.
4. N. Dragoni, F. Massacci, K. Naliuka, and I. Siahaan. Security-by-contract: Toward a semantics for digital signatures on mobil code. 2007.
5. Ú Erlingsson. *The Inlined Reference Monitor Approach to Security Policy Enforcement*. PhD thesis, Cornell University, 2004.
6. U. Erlingsson and F. B. Schneider. IRM Enforcement of Java Stack Inspection. In *Proc. of Symp. on Sec. and Privacy*, 2000.
7. David Evans and Andrew Twyman. Flexible policy-directed code safety. In *Proc. of Symp. on Sec. and Privacy*, pages 32–45, 1999.
8. Greg Goth. The ins and outs of it outsourcing. *IT Professional*, 1(1):11–14, 1999.
9. K.W. Hamlen, G. Morrisett, and F.B. Schneider. Certified in-lined reference monitoring on .net. In *Proc. of the 2006 workshop on Prog. Lang. and Analysis for Security*, pages 7–16, 2006.
10. Charles Handy. Trust and the virtual organization. *Harvard Business Review*, 73(3):40–50, 1995.
11. M. Howard and D. Leblanc. *Writing Secure Code*. Microsoft Press, 2003.
12. Y. Karabulut, F. Kerschbaum, F. Massacci, P. Robinson, and A. Yautsiukhin. Security and trust in it business outsourcing: a manifesto. In *Proc. of STM-06*, volume 179, pages 47–58. ENTCS, 2007.
13. G. Karjoth, B. Pfitzmann, M. Schunter, and M. Waidner. Service-oriented assurance comprehensive security by explicit assurances. In *Proc. of QoP-05*. Springer, 2004.
14. Brian LaMacchia and Sebastian Lange. *.NET Framework security*. Addison Wesley, 2002.
15. L.Gong and G.Ellison. *Inside Java(TM) 2 Platform Security: Architecture, API Design, and Implementation*. Pearson Education, 2003.
16. F. Massacci and K. Naliuka. Multi-session security monitoring for mobile code. Technical Report DIT-06-067, UNITN, 2006.
17. F. Massacci and I. Siahaan. Matching midlet's security claims with a platform security policy using automata modulo theory. In *Proc. of The 12th Nordic Workshop on Secure IT Systems (NordSec'07)*, 2007.
18. Ian Morris. Channel 4's 4od: Tv on demand, at a price. *CNET Networks Crave Webzine*, 2007.
19. G.C. Necula. Proof-carrying code. In *Proc. of the 23rd ACM SIGPLAN-SIGACT Symp. on Princ. of Prog. Lang.*, pages 106–119, 1997.
20. George C. Necula and Peter Lee. The design and implementation of a certifying compiler. In *Proc. of the ACM SIGPLAN 1998 Conf. on Prog. Lang. Design and Implementation*, number 5, pages 333–344, 1998.
21. Bill Ray. Symbian signing is no protection from spyware. Available at `http://www.theregister.co.uk/2007/05/23/symbian_signed_spyware/`, 2007.
22. R.Kazman, M.Klein, and P. Clements. Atam: Method for architecture evaluation. Technical Report CMU/SEI-2000-TR-004, CMU/SEI, 2000.

23. S3MS. Security of software and services for mobile systems. `http://www.s3ms.org/`, 2007.
24. F.B. Schneider. Enforceable security policies. *J. of the ACM*, 3(1):30–50, 2000.
25. R. Sekar, V.N. Venkatakrishnan, S. Basu, S. Bhatkar, and D.C. DuVarney. Model-carrying code: a practical approach for safe execution of untrusted applications. In *Proc. of the 19th ACM Symp. on Operating Sys. Princ.*, pages 15–28, 2003.
26. D. Vanoverberghe, F. Piessens, T. Quillinan, F. Martinelli, and P. Mori. Run-time compliance state of the art. Public Deliverable of EU Research project D4.1.0/D4.2.0, S3MS- Security of Software and Services for Mobile Systems, November 2006.
27. Dries Vanoverberghe and Frank Piessens. A caller-side inline reference monitor for an object-oriented intermediate language. In *Proc. of the 10th IFIP Int. Conf. on Form. Methods for Open Object-based Distributed Systems (FMOODS'08)*, volume 5051, pages 240–258, 2008.
28. David Walker. A type system for expressive security policies. In *Symp. on Principles of Programming Languages*, pages 254–267, 2000.
29. Sachiko Yoshihama, Takeo Yoshizawa, Yuji Watanabe, Michiharu Kudoh, and Kazuko Oyanagi. Dynamic information flow control architecture for web applications. In *Computer Security - ESORICS*, pages 267–282, 2007.