

Chapter 10

Security and Privacy of Online Social Network Applications

Willem De Groef

iMinds–DistriNet, KU Leuven, Belgium

Dominique Devriese

iMinds–DistriNet, KU Leuven, Belgium

Tom Reynaert

iMinds–DistriNet, KU Leuven, Belgium

Frank Piessens

iMinds–DistriNet, KU Leuven, Belgium

ABSTRACT

An important recent innovation on social networking sites is the support for plugging in third-party social applications. Together with the ever-growing number of social network users, social applications come with privacy and security risks for those users. While basic mechanisms for isolating applications are well understood, these mechanisms fall short for social-enabled applications. It is an interesting challenge to design and develop application platforms for social networks that enable the necessary functionality of social applications without compromising both users' security and privacy. This chapter will identify and discuss the current security and privacy problems related to social applications and their platforms. Next, it will zoom in on proposals on how to address those problems.

INTRODUCTION

Today social networking sites are ubiquitous and inseparable from the digital world. They host an important part of the online communication and contain the majority of people's personal information that is available on the World Wide

Web. The monetary worth of this huge amount of information is ever increasing, resulting in mind-blowing market values.

Ever since Facebook launched their application development environment, social application platforms – together with their applications itself – are ubiquitous in the context of social network-

DOI: 10.4018/978-1-4666-3926-3.ch010

ing sites. Almost every major social networking site nowadays provides means to consult personal user data from their social graph. Third-party social applications spread through the online communities and the popularity of these social applications keeps increasing. Support for such third-party applications is an important contributor to the overall success of social network sites (Pham, 2011).

Typical for these social-aware applications, is that the code provider typically is a third stakeholder, different from the social network site and the end user. Because an application has access to social data, also the application provider itself gets access. Given the growing amount of privacy-sensitive social data on social network sites, this becomes more and more an undesirable situation.

In this chapter we will focus on these privacy and security problems in the context of *online social network applications*. In this context, third-party application are typically developed in client-side scripting languages like JavaScript – which will be executed in the user’s browser - often in combination with server-side technologies like PHP or Java – fueling the back-end part of the application.

The first objective of this chapter is to introduce the details – both the architectural and technological - of the previous and current social application platforms. The second objective is to identify three different categories based on these security/privacy related problems. The third objective is to examine what recent scientific literature tries to do in order to address those problems.

BACKGROUND

According to Facebook, people install such applications more than 20 million times per day, day after day (Facebook, 2011). However, the use of such applications comes with privacy and security risk for the social network users.

Trusting Facebook, Google, and other big social network providers to respect your privacy, is hard to avoid when using social networking sites. Trusting each third-party application developer to respect the policies, imposed by the social network providers and to respect the user’s privacy, is less justified.

An investigation conducted by the *Wall Street Journal*, published October 18, 2010, claimed that nearly every popular Facebook application was leaking – in some way or another – privacy-sensitive information to other parties, such as Internet tracking companies and advertising companies (Steel & Fowler, 2010).

The basic isolation mechanisms for applications fall short for social-enabled applications: they need more fine-grained control both of the access that these applications have to information, as well as how this information is used after access has been granted. There have been various reported incidents of information leakages by social applications. Mills (2008) report on some applications allowing to peak into the social graph because they are vulnerable for a peephole vulnerability that allowed anyone to view private information. Other applications unintentionally leak private social data – while some even do this on purpose (Steel & Fowler, 2010; Kelly, 2008).

Hence, it is an interesting software engineering challenge to design an application platform for social networks that enables the rich functionality of social applications, but mitigates both security and privacy risks as much as possible.

SECURITY AND PRIVACY OF ONLINE SOCIAL NETWORK APPLICATION PLATFORMS

This section will cover two main implementation models for social network application platforms. Next it will dig into the different security and privacy issues resulting from the design choices

made in each implementation model. Finally, it will survey three proposals for more privacy-enhanced social application platforms.

Implementation Models

There exist roughly two different architectures for the implementation of social web applications. The earliest implementation model supports deployment of applications on the social network site infrastructure itself. In this so-called *gadget paradigm*, the application is described in an XML specification and embedded within the social network site infrastructure. This XML file will be loaded, converted and displayed in the context of the social network site in order to deploy the application. In this paradigm, the social network site has total control over the application's code and inserts the social data where needed. The social network site plugs the augmented code of the application in its own code. This approach allows the social network site to exercise a reasonable level of control over the application, as it can freely modify the code.

The growing trend amongst current social network sites, however, is to roll out a different approach: the social network site offers an API to query its data over HTTP. Using protocols such as OAuth, a user can delegate access to his social data to an application provider. The social application can then query the social network API over HTTP. As a consequence of this so-called *distributed paradigm*, social applications no longer have to be integrated in the social network site infrastructure itself.

The benefit for application providers is twofold. First, developers are no longer restricted to the site-specific technologies offered by the social network site. Second, the provider has complete control over its application code. One of the benefits of this paradigm shift for the social network site is the removal of the complex conversion process for the XML specifications. However,

this second model comes with additional security and privacy risks.

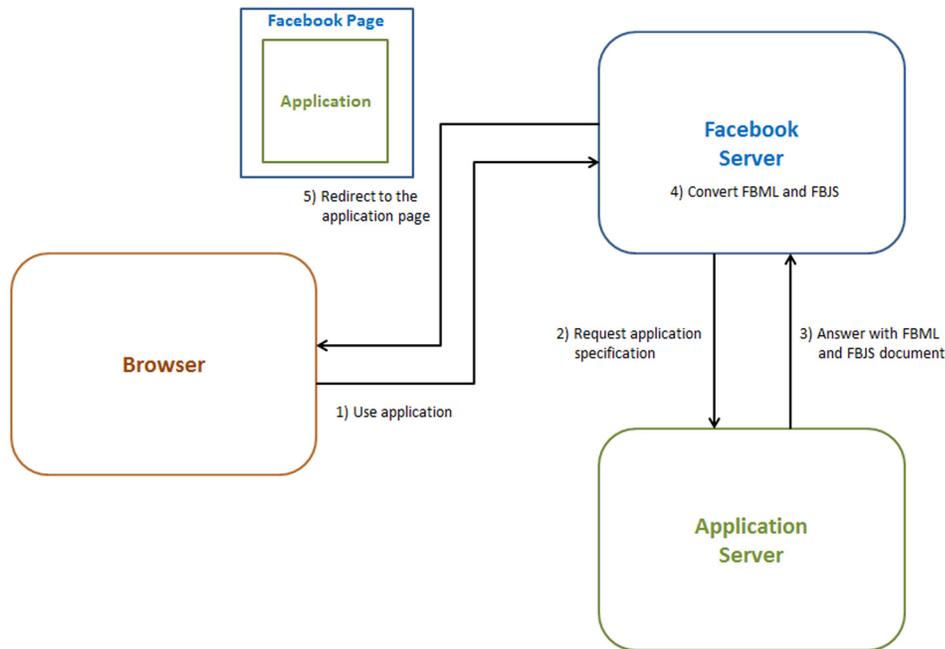
In this section of the chapter, we will describe the two main architectures for supporting social applications in more detail, as it is the crucial stepping-stone to identify the security and privacy threats.

Gadget Approach

Facebook launched their application development platform on May 24, 2007. In the months after the release, Facebook introduced the Facebook Markup Language (FBML) and Facebook JavaScript (FBJS). FBML is an extension of HTML and provides special tags to extract user-specific data from the social graph. A tag like `<fb:name uid="12345"/>` can be used to extract the name of the user with id 12345 from the social graph and embed it within the HTML page rendered in the browser. FBJS is a limited, safe subset of standard JavaScript. Facebook can rewrite this subset of JavaScript in such a way that it is safe to embed the application directly into the wrapping Facebook page, without the risk of the wrapping page being compromised by the application's JavaScript e.g., by leaking private information embedded within the page. With these two programming tools in hand, developers could start to design third-party applications for Facebook.

In this setting, applications are in fact simple documents – containing both FBML and FBJS – that have to be transmitted to Facebook in order to become a real social application. When a user wants to use an application, she browses to the application page inside Facebook. Facebook will then translate the FBML tags of the application and convert the FBJS to regular JavaScript. The result is a combination of regular HTML and JavaScript that is embedded in the corresponding Facebook page. Figure 1 illustrates the working of the original FBML/FBJS approach of Facebook applications.

Figure 1. Architectural overview of the gadget approach using FBML and FBJS



Recently, Facebook decided to change the strategy behind their application platform (Beard, 2010). They chose to shift their focus from the rather static FBML/FBJS to a more distributed approach, in order to give developers the opportunity to create more dynamic applications using regular JavaScript and HTML instead of FBJS and FBML. In April 2010, Facebook launched their Graph API, a RESTful API to extract data from the social graph. The strategy behind this RESTful concept is that it works with simple HTTP requests towards the Facebook servers, based on a unique social-entity-id. Facebook responds with the desired information in the accessible JSON¹ format.

This new strategy allows application developers to design applications that are less interleaved with the Facebook platform internals. Applications can get the desired social information directly from the server. Thus, Facebook is no longer needed as a proxy between the application server and the

user, injecting the social context into the application page on the fly.

In a more distributed approach, the application is typically hosted on a server under its own domain name. When the application is accessed as a page in the context of Facebook, this domain is loaded inside an iframe in the Facebook page. This separates the DOM of the wrapping Facebook page from the DOM of the application page, providing enough protection for the sensitive Facebook data.

Together with the introduction of the distributed approach, Facebook made its authorization policy more fine-grained. In the early days, users had only two privacy options for applications: allow the application to access all personal data, or deny it to use any data. Since April 2010, applications have to ask different kinds of permissions from the user, to access different kinds of information. This approach provides more transparency, and allows the user to make a better, well-consistent decision.

Unlike Facebook, other important social networking sites, such as Google, MySpace, Yahoo and Netlog, united their efforts to provide a uniform interoperable social application platform. The specifications of this platform were called OpenSocial and were launched in November 2007. At first sight, the OpenSocial standards are a promising competitor for the aforementioned Facebook platform.

An OpenSocial platform is quite similar to the Facebook platform as it also supports two different strategies to develop applications interacting with a social network, although less separated than those of the Facebook platform. The first strategy is a rather static approach in which the application is provided in an XML file as illustrated in Figure 2. This is called the gadget setting of an application. The XML file is converted and embedded in a wrapping page from the social networking site, similar to Facebook's FBML/FBJS approach. In this setting, applications can make calls to the wrapping social network using a JavaScript API. This strategy of directly embedding applications in a wrapping social networking site page poses the same security problems as in the Facebook platform. For this reason, the OpenSocial gadget strategy is often combined with the use of Caja (Capabilities JavaScript)², which is used in the same way as FBJS on the Facebook Platform.

As a second strategy, OpenSocial also provides a RESTful API to communicate with the social networking site. Under this model, applications operate under a separate domain or are contained in an iframe inside the social networking site. It is completely similar to the working of Facebook's Graph API in combination with the iframe setting.

Gadgets are the oldest form of embedded third-party applications in websites. As Facebook is deprecating the gadget approach, our focus will be more on the OpenSocial specification of gadgets. The general idea behind this approach is not complex. The application developer makes his application available somewhere online in a document version. Figure 2 shows how this is typically done using the XML format. In this document, the application developer provides a set of preferences and meta-information on how to run the application. As we can expect, the largest part of the document consists of the application's HTML (or FBML) and limited JavaScript.

The social networking site transforms this document later on into an application that is embedded inside the wrapping page.

Now imagine that we want to add some social context to our application. We could, as an example, change the code in Figure 2 from a simple gadget into a simple social gadget, by making the greeting personal. The corresponding

Figure 2. XML specification of a basic OpenSocial gadget

```
<Module>
  <ModulePrefs title="Hello World!">
    <Require feature="opensocial-0.8" />
  </ModulePrefs>
  <Content type="html" view="canvas">
    <![CDATA[
      <!-- HTML and JS content be here, just as if this were the
      <body> content of a web site -->
      <script type="text/javascript">alert('\Hello world!')</script>
      <h2>Hello world!</h2>
    ]]>
  </Content>
</Module>
```

XML file in Figure 3 shows that it is enough to use JavaScript code to request the nickname of the current viewer. The JavaScript API provides much more social data requests besides the one illustrated in the example. The OpenSocial JS API supports also these JavaScript API calls. All OpenSocial implementing partners are supposed to implement the OpenSocial APIs. Hence the XML specification of the very basic social application in Figure 3 is portable across all OpenSocial supporting platforms. A developer can specify one application in the OpenSocial XML format, post it somewhere online and point to it from different OpenSocial supporting social networking sites.

It is important to notice that in the gadget setting, there is no need for user authentication: because the user is already logged in with the social networking site and the application is running directly inside the page of the same social networking site, the user's identity and information can easily be transmitted to the application, without the need of extra user-authentication. It is the responsibility of the social networking site to ask the user to explicitly allow the application to access her personal data.

Distributed Approach

The distributed approach is a newer strategy to provide the necessary framework for third-party applications. In this approach, the developer hosts the application under its own domain name. Whenever the application is accessed inside the context of the social networking site, it is simply loaded into an iframe inside that social networking site and its way of operation doesn't change a bit. To support the social setting of this kind of applications that have no direct connection with the social networking site, the platform typically provides REST APIs. These APIs provide a HTTP interface to access social data. The REST-requests can be issued from the application server, from a browser or from anywhere else. Facebook and OpenSocial call their REST API, respectively, Graph API and OpenSocial REST API.

In contrast to the gadget approach in which no explicit authentication and authorization from the user are needed because the application runs inside the social networking site, the distributed approach requires an authentication procedure. Both Facebook and OpenSocial use the OAuth2.0 protocol to authenticate a user. This procedure allows a

Figure 3. XML specification of an OpenSocial Gadget

```
<Module> <ModulePrefs title="Hello World!">
  <Require feature="opensocial-0.8" />
</ModulePrefs> <Content type="html">
<![CDATA[
  <script type="text/javascript">
    function onViewerInfoLoaded(response) {
      alert('Welcome ' +
        response.get('viewerInfo').getData().getField('nickname'));}
    function loadViewerInfo(){
      var req = opensocial.newDataRequest();var extraParams = {};
      extraParams[opensocial.DataRequest.PeopleRequestFields.PROFILE_
        DETAILS] = ['nickname', opensocial.Person.Field.THUMBNAI_URL];
      req.add(req.newFetchPersonRequest (opensocial.IdSpec.PersonId.
        VIEWER, extraParams), "viewerInfo");
      req.send(onViewerInfoLoaded);}
      gadgets.util.registerOnLoadHandler(loadViewerInfo);</script> ]]>
</Content> </Module>
```

user to give the application access to her private data without having to hand out her credentials. Facebook adds a fine-grained permission model on top of the basic user authentication. We give the authentication and authorization procedure for the Facebook platform.

- The user loads the homepage of the application in his browser either directly under its domain name or inside an iframe in the context of the social networking site.
- The starting page of the application contains JavaScript code to guide the user through the authentication and authorization procedure with the social networking site. This can be done by automatically redirecting the user's browser to a URL like `https://www.facebook.com/dialog/oauth?client_id=YOUR_APP_ID&redirect_uri=YOUR_URL&scope=email,friends_likes&response_type=token`.
- If the user is not yet logged in to Facebook, he logs in.
- Then he grants the requested access rights to the application.
- Facebook generates an access token for this user and this application, appends this token to the `redirect_url` and redirects the browser back to the application `YOUR_URL#access_token`.
- The application server saves the access token, and redirects the browser to the starting page of the application.

With this access token the application can start to issue the requests for social data to the social application platform. After successfully finishing the OAuth2.0 procedure, the following request, `GET https://graph.facebook.com/me/email?&access_token=access_token` will return the e-mail address of the user who did the authorization procedure. The OpenSocial procedure is similar, apart from a far more coarse-grained permission model. In Facebook there are many

different permissions, providing the user with a certain transparency. OpenSocial only provides one specific permission: allow or deny the application to access all of your data.

Issuing these REST-requests with JavaScript from within the browser, poses a problem concerning the Same-Origin Policy of the browser. A certain origin – a triplet consisting of domain name, port, and protocol – is not allowed to access data from another origin. For example, the application running under the domain `www.yourapplication.com` will not be allowed to directly access data coming from `www.socialnetworkingsite.com`. The use of JSONP³, a standard technique to circumvent the Same Origin Policy in many situations, solves this problem.

Security and Privacy Issues

The current state of the art design of social application platforms raises several risks concerning the user's privacy-sensitive information when using third-party applications. This section addresses some of the most important issues concerning these privacy risks, both on architectural and on implementation level. As the distributed approach gains popularity and even seems to replace the gadget approach, we mainly focus on the distributed approach for the rest of this chapter.

Due to the growing popularity of social application platforms and the growing support for APIs to access social data, addressing the security issues of exposing social data to third party applications becomes important. A key problem is the total loss of control by the social network site whenever sensitive information reaches the application provider. Once a social application obtains sensitive information, it is impossible for the social network site to revoke access to information or to enforce any restrictions on where the information can or may flow to. As evidenced by Steel & Fowler (2010), Mills (2008), and Spencer (2008), this problem has become a real-world threat for social application users.

In this section of the chapter we will illustrate these security issues present in the two common implementation models for social applications discussed in the previous section.

Privacy Protection Policy

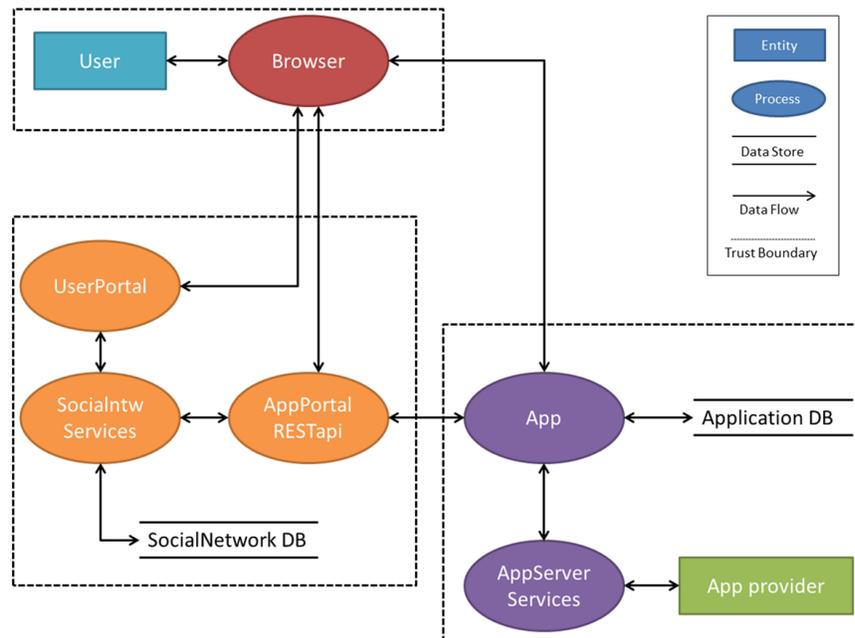
In the original privacy policy of almost all social networking sites, an application has the same view of the social network as its users. Under the pressure of the public opinion, some big social networking sites like Facebook and Google+ were forced to improve their original privacy policy. Nowadays, social networking sites tend to ask the user to agree with giving the application permission to access her data and put the responsibility of privacy protection in the hands of the end user. This strategy legally protects the social networking sites, as the user explicitly shares her data with the application. However, it can be hard for users to make a well-considered decision, especially since there is a lot of social pressure involved with using social applications.

It is important to note that OpenSocial still doesn't work with a fine-grained permission model, neither for their gadget approach nor their distributed approach, but implementers could always decide to enhance this model within their implementation.

Leaking of Social Data

The first important consequence of the current architecture of social network platforms is the fact that once the sensitive data has reached the application server, neither the user nor the social networking site can control the flow of the data anymore (see Figure 4). This is because of the fact that the application itself is running on servers out-of-control of the social networking site. The application providers have total control of all incoming information – received because the application's user allowed it. Although the application developer is legally prohibited to pass the sensitive data to third parties, there are currently no technically mechanisms to enforce this

Figure 4. Distributed social application platform design



partly because the source code of the application is mainly invisible for the end-user.

A *Wall Street Journal* investigation claimed that nearly every popular Facebook application leaked, in some way or another, sensitive private information to other parties, such as Internet tracking companies and advertising companies (Steel & Fowler, 2010). This transfer of personal information happened without the knowledge of the users. Of course, the companies behind the applications were quick to minimize the impact. They promised to fix the bug that caused the privacy breaches and after a short time of quarantine their applications were still running.

The article showed that most application developers do not hesitate to make money by selling the private information of their users. The article also showed that it is extremely difficult to verify whether every application sticks to the privacy policies - let alone to legally pursue the developers behind it in case of fraudulent behavior. Finally, it also showed that social networking sites and country laws imposing rules and policies upon third-parties are a good way to protect the privacy when there are adequate control mechanisms to verify their compliance with those rules, which is definitely not the case for the current architecture.

Issues with Access Tokens

In the current distributed design of social application platforms, access tokens play an important role in protecting the user's privacy. With a valid access token, any third-party can authenticate a request. It can even spoof that a request was allowed from a specific user. A crucial factor in the use of these access tokens appears to be their lifetime.

Facebook Access Tokens

As described earlier, a Facebook access token is the result of a successful OAuth2.0 user authentication process. Facebook keeps an access token valid for about 60 minutes. After they

expire, the application has to redirect the user back to the OAuth2.0 authentication procedure to renew the access token. For convenience, Facebook remembers that the user already granted a certain set of permissions to the application and the user's permission is given implicitly when authenticating. Facebook remembers this set of granted permissions as long as the application is present in the set of applications in the profile of the user. Thus, only when a user runs an application for the first time (or after she removed the application from her applications list), she is required to grant explicitly her permissions; all consecutive times, Facebook will implicitly grant the same permissions.

In order to protect the privacy of Facebook's users, RESTful data requests are only answered when the user is – at the same time of the request – logged in to Facebook. Obviously, Facebook also requires the user to be logged in during any explicit or implicit authentication procedure. So in theory, an application has access to the user's data during the time that the user is logged in to Facebook and until at most one hour after she stopped using the application. After that time, the application would need to renew the access token, however to obtain this token it would have to redirect the browser of the user back to the OAuth2.0 endpoint of the Facebook platform, and this is not possible. Because the user has stopped running the application, code is no longer present in the browser and the application has no longer the possibility of redirecting the user.

Strangely, Facebook offers one violation of this principle. Let us recall the scope parameter in the example of the Facebook OAuth2.0 procedure. When an application developer puts the *offline_access* permission in scope and the user grants this permission, the access token doesn't expire and the data can be accessed even when the user is not logged in to Facebook. This means that her data is accessible for the application provider, any time from now on until she actively removes the application from her profile page.

OpenSocial Access Tokens

An OpenSocial access token, by default, doesn't come with an expiration date. It is up to the implementing OpenSocial container to manage the validity of access tokens. However, OpenSocial standards provide a way to refresh the access token. The application simply has to send the longer-life refresh token that comes together with an access token. The container will respond with a new short-life access token.

OpenSocial's mechanisms to protect the privacy of users are much weaker. The OpenSocial framework doesn't provide any granularity for the permissions included in its access token. It is up to the implementing containers to manage different granularities of permissions. In practice this means that OpenSocial applications often have access to the same kind of data that the user has access to. This contains the user's private data and all the data that other people share with the user.

In contrast to the Facebook platform, where a user always has to go through an explicit or implicit OAuth2.0 procedure before the application acquires an access token, OpenSocial has an alternative built in to circumvent this OAuth2.0 procedure. OpenSocial provides code for getting the security token from inside the gadget setting without any user interaction. This means that an OpenSocial gadget can obtain an access token, even without directing a user through the OAuth2.0 procedure. This can be dangerous because the access token provides the opportunity to access all private information of the user, as we discussed before. First of all, the user doesn't know that she provided the gadget with an access token that gives access to her complete profile. On top of that, there isn't a proper mechanism to revoke the access that is given to the third parties holding this access token.

There can be another privacy problem in an OpenSocial implementation: let's assume that an access token has a lifetime of 20 minutes after the last time it was used. This means that by generating

dummy requests every 15 minutes an application developer can keep this token valid for eternity. This makes things worse – in contrast to the Facebook platform – because a user doesn't have to be online to get the answer to a REST-request from the OpenSocial platform. Finally wrapping everything together, this means that once you played a game, the application developer behind this game possibly has an access token that provides access to all of your social data. The access token can be used indefinitely and there is no clear way to revoke it.

Conclusion

We can conclude that the current design of social application platforms is far from privacy friendly. Personal sensitive information can leak away through third-party applications without the user knowing when and to whom this information is sent. On top of that it is not always very clear how to revoke the access rights once granted to third-party applications. The situation is even worse for static data like an e-mail address. Once the user granted permission to an application to read her e-mail address, there is nothing the user or her favorite social networking site can do to restrict its further spreading over the Internet.

As opposed to Facebook – that performed some updates of its platform under pressure of its users to provide a certain legally justified user privacy towards third-party applications – OpenSocial has not yet included these privacy improvements in their design.

Privacy-Enhanced Social Application Platforms

Although the subject is on the verge of scientific study, there are already some suggestions for privacy-enhanced social application platforms in the academic literature. This last section summarizes these different strategies that partially solve the privacy issues discussed above.

This section will survey and compare three different approaches. A first approach was to anonymize social data before handing it over to the application provider. Another idea was to apply concepts from information flow analysis on social applications to contain and regulate information flowing in and out of the application. The third we will discuss is a countermeasure to prevent access tokens from ever leaving the browser.

Privacy-by-Proxy

Already in the early days of the integration of third-party applications with social networking sites, Felt and Evans (2008) uncovered the privacy problems concerning these applications and came up with a solution. As their work already got published in 2008, they based their research on the first Facebook approach, the FBML/FBJS strategy. Although the limitation of their solution to the gadget approach is currently outdated, it was justified in 2008 and is therefore worth to be studied. Being the first to address privacy problems with social application platforms, they did pioneering work both in studying the state of the art in those days and in proposing a solution for the problem. They focused in their work on the Facebook approach, although they argued that it would also work in the OpenSocial gadget approach.

Felt and Evans (2008) propose an implementation for a privacy-enhanced social application platform based on what they call *privacy-by-proxy*. In the gadget approach, the social networking site always hosts the application inside its own page, and it performs a translation from FBML and FBJS to regular HTML and JavaScript. Hence the social networking site acts as a proxy between the application's code and the visualization towards the user.

Felt and Evans (2008) propose to encrypt user ids in order to anonymize the contents of the social graph. The applications are only allowed to work with this anonymized data. They also

extend the markup language to include tags that allow working with these anonymized ids. For example, `<uval id="[id]" field="birthday" />` would put the birthday of the user on the screen, based on her anonymized id. Upon the request for the friends of a certain user, an application gets a list of anonymized ids. This way an application can only work with an anonymized social graph. It is very important that de-anonymized data doesn't return to the application server. Because their solution works in the gadget approach, the social networking site has the possibility to statically check – i.e., before it has the chance to get executed – whether the contained JavaScript (FBJS) doesn't leak any data.

This implementation needs one more element to make it completely secure. There has to be a limitation in the set of tags that are translated in the application of a certain user. It is necessary to de-anonymize the birthday of a friend. However, the social networking site should not de-anonymize the birthday of a stranger, because in that case an evil application developer could install its own application and create an extra page in which he exhaustively de-anonymizes his whole database. Felt and Evans (2008) solve this problem by only de-anonymizing data from users that appear on a certain contact list. This list summarizes all people with whom the user has had some sort of interaction.

Felt and Evans (2008) tested their implementation, and only a small minority of the popular Facebook applications of those days would stop working. This proved that the approach was both secure and maintained the important functionality of third-party applications. However, since Facebook and OpenSocial introduced their distributed approach, the social networking site lost its role as proxy between the application and the user. Hence it can no longer translate the tags in a privacy concise way or prevent the private data from being leaked to the application server, as the application server can request personal data directly.

xBook

Singh, Bhola, and Lee (2009) came up with a different, more complex solution addressing the privacy issues concerning social third-party applications – called xBook. It is an architectural framework that combines a social networking site with untrusted third-party applications. It provides a container in which the third-party applications are deployed. That way the xBook framework can monitor and regulate all information streams that pass through the third-party application.

As xBook is a complete architecture that contains both the client side and server side implementation of the third-party applications, it has the possibility to impose strong regulation mechanisms. In order to do this, third-party applications are split up into different components. The subdivision of the components is based on their interaction with third parties and handling of private data. For example: one client side component renders the application's pages in the web browser together with the privacy sensitive data; while another client side component is in charge of communicating with a third-party server. In this setting the second component doesn't have access to the private information contained in the first component. The only way that components can send data to each other is by using xBook as a mediator.

Upon installation of the application, a manifest is presented to the user stating which data will be accessed and shared with which third parties. During operation, xBook verifies that the different components – forming the application – keep to the policy. This strategy gives xBook a powerful mechanism to regulate information streams and make data transfers explicit to the user. The same ideas are implemented for the server side components as well. xBook mediates all inter-component communication and because each component has its own responsibility, it knows where and how private data leaks out of the system.

xBook works in a very rigid way: it addresses all the privacy related issues, discussed in the first part of this chapter. However, the framework comes with some disadvantages. First of all, it is a complex system. It has to statically or dynamically check whether the components behave like they are assumed to behave. For example, a component that has access to private information cannot make calls to third-party servers. This strategy involves the use of JavaScript libraries like Caja or alike for the client side components, and statically checking its compliance. It also requires a server side labeling method to prevent that data leaks unnoticed out of the framework. On top of all these control mechanisms the application server components also run inside of the social networking site's framework, consuming server resources. Finally, even when a social networking site decides to provide the servers and implement all the control mechanisms, it is very unlikely that third-party applications will be happy to provide all their code, even that of the server side, to the social network.

We can conclude that xBook is a theoretically excellent solution for the privacy issues addressed in this chapter. However, disadvantages of xBook are its complexity and possibly the unwillingness of third-party developers to make applications for the framework.

PoX

The third example of a privacy-enhanced social application platform that can be found in the literature is called PoX (Egele, Moser, Kruegel, & Kirda, 2011). It is based on a client-side-proxy idea. The design focuses on providing a privacy-secure environment when using the applications in the distributed approach. The fact that applications can request privacy-sensitive information from the social networking site without the users even being aware of it, poses a major privacy issue in the distributed setting. The third-party application

only needs the right access token to get access to the desired information.

PoX introduces a client-side-proxy that mitigates this kind of unnoticed data exchanges by preventing that the access token ever leaves the browser. Without this access token, no third-party can get private data from the social networking site; hence all data exchanges have to pass through the client-side-proxy in the browser. Applications have to request data to the client-side-proxy in the browser of the user. Based on policies – set by the client in an access control list (ACL) – PoX decides whether to forward the request to the social networking site and send the data back to the application server or whether it ignores the request. In this design the user herself can monitor and regulate which personal data are transmitted to third-party applications and which data have to remain confident.

In detail, PoX consists of (1) a plug-in for the browser that filters the access token from the HTTP stream and (2) some code running inside a hidden iframe in the browser. This code uses a technique known as ‘long polling’ to configure the server-to-client communication. When the application is loaded, the PoX code that resides in the browser makes an HTTP request to the application server and tries to fetch a kind of dynamic webpage. As long as the application doesn’t need information, it stalls the response to this request. Once it needs extra personal information, the server generates a response containing the data request. The PoX code decides whether it should forward the request. If yes, it forwards the social networking site’s answer to the request and the system goes back into the stalling modus.

In contrast to xBook, PoX is a very lightweight privacy protection mechanism. Despite this relatively simple design it reaches a lot of its goals. With PoX, a user can manage her privacy settings according to her own wishes, and enforce on her own that those wishes are respected. However, both xBook and PoX suffer from the same problem, as the user can still be misled by a third-party

application. Imagine for example the case in which a user makes a mistake and trusts an evil third-party application, and she grants access to her personal data. In both designs the third-party application can send this sensitive data to another third-party server out of the control boundary of xBook and PoX. Because of this one-time mistake, the user has lost her information and has no way of revoking the access rights of that information.

FUTURE RESEARCH DIRECTIONS

The three examples found in literature and described earlier, each provide a different interesting point of view on the subject. However, none of them completely solves the earlier listed issues. The approach of Felt and Evans (2008) shows some interesting aspects of anonymization, but unfortunately the social application platforms have evolved a lot since the publication: the solution doesn’t work anymore for the distributed approach. Both xBook and PoX map better on the current state-of-the-art social application platforms, but they do not prevent that social data leaks away to servers beyond the control of the user or the social networking site. These findings suggest that there is no privacy enhanced social application platform that maps on the current state of the art and still prevents sensitive data to leak away.

In the distributed approach to social application platforms, maintaining privacy guarantees is essentially an information flow problem (Sabelfeld & Myers, 2003). Third party code gets access to sensitive information and should be prevented from leaking that information to inappropriate channels.

Fortunately, the information flow security community has made significant steps forward over the past decade. In particular, several static and dynamic approaches to enforce information flow security in JavaScript-like languages have been proposed (Hedin & Sabelfeld, 2012; Hedin & Sabelfeld, 2012; Austin & Flanagan, 2012; Chugh, Meister, Jhala, & Lerner, 2009; Bielova, Devriese,

Massaci, & Piessens, 2011; Devriese & Piessens, 2010). Hence, we believe that a promising avenue for future work is the application of these enforcement mechanisms to social applications.

CONCLUSION

The two largest social application platforms – the Facebook platform and the OpenSocial standards – both support the distributed approach to integrate third-party applications within their framework. The OpenSocial gadget approach poses some serious security and privacy risks, so it is questionable whether it will be around for a long time. A support for this claim can be found in the deprecation of Facebook’s gadget approach in January 2012.

Apart from the privacy violations identified in the OpenSocial gadget approach, these social application platforms are far from being privacy-friendly. Third-party applications typically have far more access to the user’s social data than strictly needed to function, effectively violating the principle of least-privilege and posing privacy threats.

Several privacy-enhanced social application platforms are developed, but none of them seems to completely solve the issues for the most common distributed approach. A promising future direction is to develop such a privacy-enhanced platform using novel enforcement mechanisms recently developed in the information flow security community.

ACKNOWLEDGMENT

This research is partially funded by the Research Fund KU Leuven, the EU-funded FP7 projects NESSoS and WebSand and by the IWT-SBO project SPION. Dominique Devriese holds a Ph.D. fellowship of the Research Foundation – Flanders (FWO).

REFERENCES

- Austin, T. H., & Flanagan, C. (2012). Multiple facets for dynamic information flow. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*.
- Beard, E. (2010, April 21). *A new data model*. Retrieved October 6, 2012, from <http://developers.facebook.com/blog/post/2010/04/21/a-new-data-model/>
- Bielova, N., Devriese, D., Massaci, F., & Piessens, F. (2011). Reactive non-interference for a browser model. In *Proceedings of the International Conference on Network and System Security (NSS)*.
- Chugh, R., Meister, J. A., Jhala, R., & Lerner, S. (2009). Staged information flow for JavaScript. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)* (pp. 50-62).
- Devriese, D., & Piessens, F. (2010). Noninterference through secure multi-execution. In *Proceedings of the IEEE Symposium on Security and Privacy* (pp. 109-124).
- Egele, M., Moser, A., Kruegel, C., & Kirda, E. (2011). PoX: Protecting users from malicious Facebook applications. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM)*.
- Felt, A., & Evans, D. (2008). Privacy protection for social networking platforms. In *Proceedings of the Workshop on Web 2.0 Security and Privacy (W2SP)*.
- Hedin, D., & Sabelfeld, A. (2012). Information-flow security for a core of JavaScript. In *Proceedings of the IEEE Computer Security Foundations Symposium (CSF)*.
- Mills, E. (2008, June 26). *Facebook suspends app that permitted peephole*. Retrieved October 6, 2012, from http://news.cnet.com/8301-10784_3-9977762-7.html

Pham, A. (2011, September 9). Article. *LA Times*. Retrieved October 6, 2012, from <http://latimesblogs.latimes.com/entertainmentnews-buzz/2011/09/sims-social-surpasses-farmville-as-second-largest-facebook-game.html>

Sabelfeld, A., & Myers, A. C. (2003). Language-based information-flow security. *IEEE Journal on Selected Areas in Communications (JASC)*, 5–19. doi:10.1109/JSAC.2002.806121

Singh, K., Bhola, S., & Lee, W. (2009). xBook: Redesigning privacy control in social networking platforms. In *Proceedings of the USENIX Security Symposium* (pp. 249-266).

Spencer, K. (2008, May 1). *Identity 'at risk' on Facebook*. Retrieved October 6, 2012, from http://news.bbc.co.uk/2/hi/programmes/click_online/7375772.stm

Steel, E., & Fowler, G. A. (2010, October 18). Facebook in privacy breach: Top-ranked applications transmit personal IDs, a journal investigation finds. *The Wall Street Journal*. Retrieved October 6, 2012, from <http://online.wsj.com/article/SB10001424052702304772804575558484075236968.html>

ADDITIONAL READING

Fogie, S., Grossman, J., Hansen, R., Rager, A., & Petkov, P. D. (2007). *XSS attacks: Cross site scripting exploits and defense*. Syngress.

Hawker, M. D. (2010). *The developer's guide to social programming: Building social context using Facebook, Google Friend Connect, and the Twitter API*. Addison-Wesley Professional.

LeBlanc, J. (2011). *Programming social applications: Building viral experiences with OpenSocial, OAuth, OpenID, and distributed web frameworks*. O'Reilly Media.

Zalewski, M. (2011). *The tangled web*. No Starch Press.

KEY TERMS AND DEFINITIONS

Application Provider: The stakeholder that provides the software of an application, as opposed to other stakeholders such as the end-users of the software or the owner of the infrastructure that runs the software.

Gadget: In the context of this chapter, a gadget is a piece of software that runs in the context of an online social network, and is hosted on the online social network infrastructure.

Information Flow Analysis: The analysis of how inputs to a program influence the outputs of that program, in order to put bounds on the information that can be deduced about private inputs by observers that can only observe public outputs.

JavaScript: A prototype-based dynamic scripting language originally developed by Netscape to support scripting of their web browser. JavaScript was later standardized as ECMAScript, and all major web browsers today support a variant of the language.

Online: Social Network (OSN): An online platform where users can publish public or semi-public profile information, form relationships with other users, and interact in a variety of ways with these relations and with the general public.

Representational State Transfer (REST): An architectural style for distributed software systems that emphasizes uniform, stateless and cacheable software interfaces.

Social Application: In the context of this chapter, a social application is a third-party provided software application that runs in the context of, or in interaction with an online social network, and that makes use of the profile and relationship information that is present in that online social network.

ENDNOTES

- ¹ JSON (JavaScript Object Notation) is a textual representation – based on open standards – of JavaScript objects, designed for human-readable data interchange across the Internet.
- ² Caja is a technology to automatically convert JavaScript into a safe subset, based on the

concepts of object-capability. The technology is primarily used to run third-party JavaScript code within a sandbox.

- ³ JSONP is a method to request data from a cross-origin domain. It is primarily used as a browser-independent method to circumvent the same-origin policy.