Designing malicious applications in social networks

Sotiris Ioannidis

Designing Malicious Applications in Social Networks

Andreas Makridakis, Elias Athanasopoulos, Spiros Antonatos, Demetres Antoniades, Sotiris Ioannidis, and Evangelos P. Markatos
Institute of Computer Science (ICS)
Foundation for Research & Technology Hellas (FORTH)
{amakrid,elathan,antonat,danton,sotiris,markatos}@ics.forth.gr

Abstract

The World Wide Web has evolved from a collection of static HTML pages to an assortment of Web 2.0 applications. Online social networking in particular is becoming more popular by the day since the establishment of SixDegrees in 1997. Millions of people use social networking web sites daily, such as Facebook, MySpace, Orkut and LinkedIn. A side-effect of this growth, is that possible exploits can turn OSNs into platforms for malicious and illegal activities, like DDoS attacks, privacy violations, disk compromise and malware propagation.

In this paper we show that social networking web sites have the ideal properties to become attack platforms. We introduce a new term, *Antisocial Networks* that refers to distributed systems based on social networking web sites which can be exploited carry out network attacks. An adversary can take control of the visitor's session by remotely manipulating their browsers through legitimate web control functionality such as image-loading HTML tags, JavaScript instructions and Java applets.

I. Introduction

The popularity of online social networks (OSN) [28] is ever increasing. The online communities created by OSN are a fast growing phenomenon on the web empowered by new modes of social interaction among people from around the globe. OSN are useful for keeping in touch with friends, forming new contacts, research collaboration, information sharing, political campaigns [35], [39], *etc.* Some OSN are used for professional contacts, *e.g.* LinkedIn [10] and XING [16], where a user can discover business connections, while others, such as Facebook [1], MySpace [12] and Orkut [14], are friendship-focused and are primarily used for communication, photo and video sharing and entertainment.

The massive adoption of OSN by Internet users, as described above, provides us with a unique opportunity to study possible exploits that may turn them into *Antisocial Networks*, that is platforms for malicious and illegal activities, like DDoS attacks, malware propagation, spamming, privacy violations, disk compromise, to name a few. OSN have some intrinsic properties that make them ideal for exploitation by an adversary: (i) a very large and highly distributed user-base, (ii) clusters of users sharing the same social interests, developing trust relationships and seeking access to the same resources, and (iii) platform openness for deploying malicious applications that lure users to install them.

All these characteristics give adversaries the opportunity to massively manipulate Internet users and force them to perform antisocial acts against the rest of the Internet, *without* their knowledge. Apart from controlling social network users and driving them to launch attacks against third parties, an adversary can also harm the users themselves.

In this paper we explore these properties, develop real exploits and analyze their impact.

II. Facebook Architecture

Facebook, one of the leading social networks, offers a framework for software developers to create lightweight applications. These application are able to run inside the social network and interact with its resources (users and users' data). In this section we present technical details for the construction of Facebook applications.

A. Platform Overview - Core Components

A user who wants to build a Facebook application must simply add the Developer Application [3] to her account. The application can be implemented in any development enviroment the user prefers. Facebook and third party application developers have created a large number of client libraries in several development enviroments. Facebook officially supports PHP, JavaScript, Connect for iPhone and Flash/ActionScript client libraries. Facebook does not provide official support for the following client libraries: Android, ASP.NET, ASP (VBScript, JScript), Cocoa, ColdFusion, C++, C#, D, Emacs Lisp, Erlang, Google Web Toolkit, Java, Lisp, Perl, Python, Ruby on Rails, Smalltalk, Tcl, VB.NET and Windows Mobile.

For an application to work correctly with Facebook, the user needs to fulfill the following requirments:

1) Provide it access to a Web server able of hosting the application.
2) Upload the appropriate Facebook client library to the server.
3) In case the application has any storage requirments, i.e. stores user information[1], the user should also provide

[1]Storable Data: http://wiki.developers.facebook.com/index.php/Storable_Information

it access to a relational database management system, such as MySQL.

4) Register the application in Facebook.

To register the application in Facebook, the user fills out a form and submits the application, through the Developer Application. The form requires information, such as the application name, the application description, the IP address of the hosting web server, *etc.* Some critical fields are: (i) the Canvas page URL, and (ii) the Canvas callback URL. A Canvas page is the main page of an application in Facebook. When users access the application they are redirected to this URL. The actual format is `http://apps.facebook.com/canvas_page_name`, where `canvas_page_name` is usually the name of the application. The Canvas callback URL is the address of the web server or hosting service where the application lives on.

Typically, a few days after submitting the application the Facebook Platform Team notifies the developer either that the application has been successfully accepted or that it has been rejected. If the application is accepted, it is added to the Application Directory[2], that allows users to easily find new applications and install them to their profile.

Following we provide a short description of some basic components provided by Facebook for assisting application development.

Facebook Markup Language: The Facebook Markup Language (FBML) [5] is a subset of HTML along with some additional tags specific to Facebook. Specific tags have a common form: <`fb:tagName/`>. FBML lets applications interact with their user and the user's friends.

Facebook Query Language: The Facebook Query Language (FQL) [6] allows a developer to use an SQL-style interface to easily query some Facebook social data, such as the full name or profile picture of a user. The Facebook Platform provides several tables [3] as a reference for constructing FQL queries.

Facebook JavaScript: Facebook JavaScript (FBJS) [4] permits developers to use JavaScript in their applications. FBJS has the same syntax, as the traditional JavaScript. When FBJS source code gets parsed any identifiers (function and variable names) get prepended with the application's unique identifier, preventing the sandboxing of the code. Furthermore, FBJS supplies powerful AJAX object and an animation library to developers.

Facebook API: Using the Facebook API [2], a developer can add social context to their application by utilizing profile, friend, fan page, group, photo and event data. For example, through specific methods, a developer can collect user's hometown location, high school information, favorite quotes, *etc.* The API uses a REST-based interface. This means that the Facebook API method calls are made over by simply sending HTTP GET or POST requests to the Facebook API REST server. Thus allowing any programming language with HTTP support to be used for communication with the REST server and retrieve the needed social data.

[2]Facebook Application Directory: http://www.facebook.com/apps/directory.php

[3]FQL Tables: http://wiki.developers.facebook.com/index.php/FQL_Tables

B. FBML Canvas Example

When a user accesses a Canvas page, several steps occur in the Facebook REST server and the hosting server in order to render application's contents to the user's browser. These steps are depicted in Figure 1. Initially, the user's browser requests the Canvas page URL from the Facebook server. Following, the Facebook server sends an HTTP POST request to the application hosting server for the Callback URL, asking for the FBML of the Canvas page. If the application needs to retrieve any social data then the hosting server sends an HTTP GET/POST request to the Facebook REST server for the needed data. After executing all API method calls, the hosting server returns the resulted FBML to the Facebook server. The Facebook server transforms that FBML into HTML and sends it back to the user's browser. This way, the Facebook server acts as a proxy between the user's browser and the hosting server.

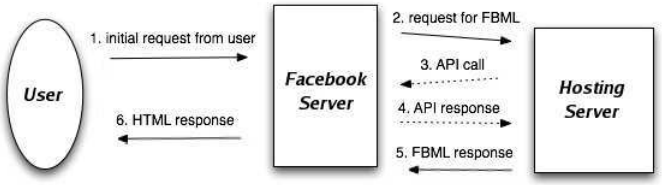

FIG. 1: How an FBML Canvas page is rendered.

III. ANTISOCIAL ACTIVITIES

This section we presents a number of proof-of-concept attacks based on Facebook applications. Specifically, we present applications that aim to launch Distributed Denial of Service (DDoS) attacks against third parties, fetching remote files from a victim's machine and harvesting sensitive personal information of users.

A. OSN as DDoS Platform

To exploit a social Web site like Facebook for launching DDoS attacks, the adversary needs to create a malicious application, which embeds URIs linking to the victim's Web server. These URIs point to documents hosted by the victim, like images, text files, media files, *etc.* When a user interacts with the application, the victim host will receive unsolicited HTTP GET requests. These requests are triggered via Facebook, since the application lives inside the social network, but they are actually generated by the Web browser of the user who access the malicious application. We define as *FaceBot* the collection of Web browsers that are forced to generate requests upon running a malicious Facebook application (see Figure 2). The cloud groups a collection of Facebook users who interact with a malicious Facebook application. This causes a series of requests to be generated, by the client, and directed towards the victim.

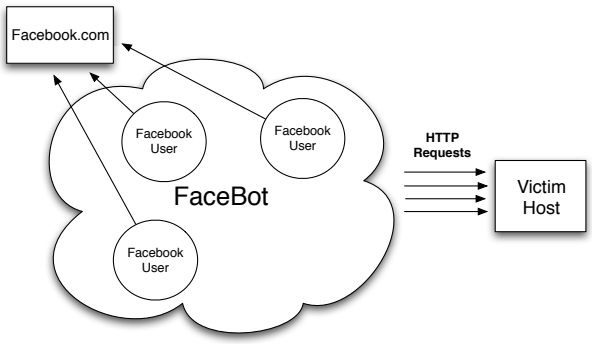

FIG. 2: The architecture of a FaceBot. Users access a malicious application in the social site (facebook.com) and subsequently a series of HTTP requests are created targeting the victim host.

We created a real-world Facebook application, called *Photo of the Day* [15], that displays a different photo every day. When a Facebook user accesses the application, they can see the photo of the day, along with a brief description below it. Clicking on the photo provides a wallpaper version of it. Lastly, users can invite their friends, who do not have the application, to add it to their profile. We do not employ any obligatory invitation for users willing to install the application. It is very common that Facebook applications require a user to invite a subset of their friends, and thus advertise the application to the Facebook community, prior the installation. This practice helps for further propagation of the application in Facebook.

Malicious Attributes: To modify *Photo of the Day* from an innocent-looking application to a malicious application we have placed special FBML tags in its source code, so that every time a user visits the application's Canvas page, HTTP requests are generated to a victim host. More precisely, the application embeds four hidden frames with inline images hosted at the victim. Each time the user interacts with the application, the inline images are fetched from the victim, causing the victim to serve a request of 600 KBytes. The user is not aware of this, since the additional images are never displayed. We list a portion of our sample source code in Figure 3.

Notice, that the application absorbs a fixed amount of traffic from the victim host. An adversary can employ more sophisticated techniques through a JavaScript program that continuously requests documents from a victim host over time. In this fashion the attack can be significantly amplified.

The hidden frames are included in the application's HTML source code via the <fb:iframe/>[4] FBML tag. The traditional <iframe/> HTML tag has been recreated by the Facebook Platform into <fb:iframe/> in FBML. The code listed in Figure 3 is generated by a Facebook server, when the latter transforms the following FBML code into HTML.

```
<fb:iframe name="1" style="width:0px; height:0px; border: 0px"
src="http://victim-host/image1.jpg"></fb:iframe><br/>
```

One may argue that invisible images can be used through

[4]FBML iframe: http://wiki.developers.facebook.com/index.php/Fb:iframe

```
<iframe name="1" style="border: 0px none #ffffff;
width: 0px; height: 0px;"
src="http://victim-host/image1.jpg?
fb_sig_in_iframe=1&amp;
fb_sig_time=1202207816.5644&amp;
fb_sig_added=1&amp;
fb_sig_user=724370938&amp;
fb_sig_profile_update_time=1199641675&amp;
fb_sig_session_key=520dabc760f374248b&amp;
fb_sig_expires=0&amp;
fb_sig_api_key=488b6da516f28bab8a5ecc558b484cd1&amp;
fb_sig=a45628e9ad73c1212aab31eed9db500a">
</iframe><br/>
```

FIG. 3: Sample code of a hidden frame, inside a Facebook application, which causes an image, namely `image1.jpg`, to be fetched from a remote host.

the common <img/> HTML tag. However, the Facebook Platform handles `img` specially. [5]. When an application is published, Facebook servers request all the image URLs from the hosting server and then serves these images, by rewriting the `src` attribute of all `img` tags using a *.facebook.com domain. This technique protects the privacy of Facebook users and does not allow malicious applications to extract information from image requests made directly from a user's browser. To overcome caching of images, by Facebook servers, we use hidden frames, which do not utilize the above caching properties.

Hosting Issues: The adversary must also provide hosting for the malicious application, must be able to cope with requests from users that are accessing the application. However, this can be avoided using a free hosting service specifically designed for Facebook applications (for example Joyent [6]). Even if such a service is not available, the adversary has to cope with significantly less traffic than the one that targets the victim.

B. Embedded Self-Signed Java Applets

The Facebook Platform gives developers the opportunity to embed a Java applet inside their applications. With this in mind we examine if we can exploit the trust among users and create malicious self-signed applets.

More specifically, the self-signed applet advises the user to upload a file, stored locally on their computer, through a common file selector. The upload capability is not possible for non-signed applets. When a user uploads the corresponding file, since they have accepted the digital signature, the Java applet has full control over the user's disk. Note, also, that the adversary has access to the user's disk even if the user selects no file.

To demonstrate the attack, we create six distinct Facebook applications that exploit the above self-signed applets' vulnerability, while at the same time not causing any harm to Facebook users who install them to their profile.

Users that access these applications have to upload a photo, which is transformed to colored HTML. Also, a random lucky

[5]Facebook Platform handles img tags in a special manner: http://wiki.developers.facebook.com/index.php/UsageNotes/Images

[6]Free Facebook Applications Developer Program: http://www.joyent.com/products/joyent-developer-programs/free-facebook-dev-program

color and number are provided, hence the term *Lucky*. Except from Lucky Art [11], our first proof-of-concept signed applet, the remaining five applications are brief quizzes, where the user has to upload a photo in order to find out the results of their answer sheet.

In order to create the certificates needed for digitally signing our applets, we used the Keytool utility [9]. Below, we list the steps needed to create the self-signed file upload applet, used by Lucky Art:

1) `keytool -genkey -keyalg rsa -alias FacebookLuckyArt`
2) `keytool -export -alias FacebookLuckyArt -file FacebookLuckyArt.crt`
3) `javac upload/FileUpload.java`
4) `jar cvfm applet.jar upload/*.class`
5) `jarsigner applet.jar FacebookLuckyArt`

The Java Archive file `applet.jar` contains the Java classes that constitute the self-signed applet. When a user interacts with the Lucky Art application, the Java Runtime Environment (JRE) displays a certificate request, stating that the application's digital signature cannot be verified, and the user has to accept the digital signature in order to run the applet.

A Java applet is included in an HTML page, by using the $<$`applet/`$>$ tag. However, this tag is not allowed in a Canvas page of a Facebook application. Developers have requested the applet tag [7], but Facebook operators have not supported it yet. Therefore, the trick to embed our self-signed applet inside Lucky Art, was to use the FBML $<$`fb:iframe/`$>$ tag. More precisely, the Canvas page contains the following FBML code:

```
<fb:iframe src="http://host-ip/lucky_art/applet.html"
name="file_upload_applet" width="100%" height="125"
scrolling="auto" frameborder="0"></fb:iframe>
```

The source of the `iframe` is an HTML page, where the applet tag is used to load the applet, thus it contains the following HTML code:

```
<html>
<body>

<applet code="upload.FileUpload.class" archive="applet.jar"
name="Lucky Art on Facebook!" width="100%" height=107>
</applet>

</body>
</html>
```

We have also experimented with The Lucky Art application in two other online social networks, `orkut.com` [14] and `friendster.com` [7].

The Developer Platform of Orkut is based on OpenSocial [13], and provides developers with a complete API to build rich social gadgets. Social gadgets are XML files, where a developer specifies the main information of their gadget, and includes all the HTML, CSS, and JavaScript source code that constitutes her application. Thus, in order to build Lucky Art on Orkut, we simply provide an XML file.

Friendster has a Developer Platform, called fbOpen Platform [8], that is based on the Facebook Platform. Thus developers can bring over applications that have been developed for Facebook and make them available to the Friendster user community. Therefore, we did not need to spend any additional resources to bring the application to Friendster.

The Friendster Platform follows exactly the same process, as the one presented for Facebook, in order to approve or not a submitted application. The first submission of Lucky Art was not approved. They informed us that: *Lucky Art cannot be added to the directory at this time for the following reasons:* (i) *Cannot install,* (ii) *Got security warning when attempting to install*. Therefore, we removed the self-signed applet from Lucky Art and keep the rest of its operation intact. After these changes we submitted Lucky Art again and after a few hours Friendster notified us that the application was successfully approved. A few days after, we restored application's functionality to the original's one, including the self-signed applet. Friendster never complained about the fundamental changes made to the application's functionality.

C. Personal Information Leakage

Facebook gives users the opportunity to have their profile locked and visible only by their friends. However, an attacker can collect sensitive personal information of Facebook users, without their permission. A Facebook application has full access in the majority of user's *and* user's friends details, by calling methods of Facebook API[8]. Although, users can set the privacy settings for each installed application, Felt *et al.* in [31] state that most users give all applications the right to have full access to their account details. An adversary could deploy an application, which simply posts all available user details to an external colluding Web server. In this way, the adversary can gain access to the personal information of users, who have installed the malicious application.

The adversary can borrow the name of an existing famous application for this purpose. For example, if a popular application is called *"I am a popular app"*, the adversary can create an application with the name *"I am a popular app II"*. Through this, the application can gain popularity among Facebook users, namely more and more user profiles would be logged to the attacker's Web server. Note that, the adversary does not need to clone the overall functionality of the existing application, but only its name. For example, the malicious application can be a random quiz. After the installation the user will realize that this application has no value, and they will probably uninstall it. Even if the user does so, the attacker has already accessed the user's personal details upon the first interaction between the victim user and the application.

To test this hypothesis, we create five distinct Facebook applications, where each of them borrows the name of an existing famous application. All these cloned applications have a brief quiz as a Canvas page. Note that during this study we do not collect any kind of personal information from users who installed the cloned applications.

[7]Requested FBML Tags: http://wiki.developers.facebook.com/index.php/Requested_FBML_Tags

[8]e.g. `Users.getInfo`: http://wiki.developers.facebook.com/index.php/Users.getInfo

Our five cloned applications are submitted to Facebook Platform as brief quizzes. After the notification that they are successfully accepted and listed in the Applications Directory, we change their name and Canvas page URL. For demonstration purposes, after the approval, we integrate the appropriate calls to API methods for extracting the personal information of our own profile only.

Facebook Platform imposes a timeout of near 10 seconds, for rendering an application page to the visiting user. More specifically, the pages that Facebook servers get from the hosting Web server should return in a timely fashion. Otherwise, users get a corresponding error. As stated above, in order to get a wide range of personal information, a developer should execute many API methods. Taking into account Figure 1, and concerning the RTT between a Facebook server and the hosting server, we can realize that a developer has to wait a lot of seconds or even minutes for receiving all the needed personal data. Indeed, when we include API calls for extracting all of our personal information, we got an error from Facebook. To overcome this, we use more `iframes` so that the basic HTML response is delivered in parallel to a Facebook server.

Towards the above solution, we include six hidden frames in the index page of our application. The first one is responsible for makng the appropriate calls to retrieve the data of the visiting user. The remaining `iframes` are responsible to retrieve user's friends personal details. Each of them, undertakes to process the 20% (1/5) of user's friends. Note that, through this approach, if a user has a short interaction with the application and then navigates away from it, the PHP code that was loaded in the hidden frames will be still running.

We list below the FBML code included in the index page, in order to embed a hidden frame which is responsible to fetch personal details of visiting user's friends.

```
<fb:iframe
src="http://apps.facebook.com/app-name/set1.php"
name="set1" width="0%" height="0%"
scrolling="no" frameborder="0">
</fb:iframe>
```

IV. Experimental Evaluation

We will now provide a brief evaluation of all the proof-of-concept attacks outlined in Section III. For the FaceBot attack a complete evaluation can be found in [23].

A. Self-Signed Applets Acceptance

For the self-signed Java applets that live on `facebook.com`, `friendster.com` and `orkut.com`, we observe whether a user accepts the digital signature or not. In Table I, we present the unique installations for our applications, Lucky Art and Facebook quizzes, and the percentage of users who accepted or denied the digital signature presented to them. We also show how many users initially denied the digital signature but then accepted it.

As we can see, eight social applications reached 1,009 installations, where 655 users have accepted the requested digital signature. Therefore it is possible to to compromise 65% of our user-base and potentially read their locally stored files without their permission. It is evident that users are willing to accept a digital signature that cannot be verified by a trusted source.

B. Popularity of Cloned Applications

In a similar fashion, in Table II, we present the installations for two cloned applications in Facebook.

Application	Installations	Profiles
Cloned Application 1	15,395	999,980
Cloned Application 2	2,531	290,137

TABLE II: Popularity of two cloned application in Facebook. Nearly 1.3 million Facebook profiles can be posted to an attacker's web server.

A large user-base has already installed the applications, so borrowing the name of an already popular application can boost a cloned application's propagation in a social network. As we can see, the personal details of nearly 1.3 million Facebook profiles could be posted to an attacker's web server.

V. Related Work

Several researchers have conducted significant work towards the structure and evolution of online social networks [22], [24], [37], but little work has been done on measuring real attacks on these sites. Apart from scattered blog entries that report isolated attacks, such as malware hosting in MySpace [17], [18], there have been no large-scale attacks using social networking sites, reported or studied so far. The most closely related work to FaceBot was done by Lam *et al.* in [36]. Our work here extends the idea of Puppetnets by taking into account the characteristics social networking web sites.

Jagatic *et al.* in [34], Hogben in [33] and Brown *et al.* in [29] study how phishing attacks [30] can be made more powerful by extracting publicly available personal information from social networks. Identifying groups of people leads to more successful phishing attacks than by simply massively sending e-mails to random people unrelated to each other. In [32], the authors examined that a large amount of students, at Carnegie Mellon University, are disclosing personal information in Facebook, not taking into account the site's privacy settings. Even if users limit their privacy preferences to allow personal information dissemination only to their friends, malicious users can use automated identity theft attacks, presented by Bilge *et al.* in [25], in order to gain access to a large volume of personal user information. Their attack method consists of cloning an existing user/victim profile and sending friend requests to her contacts. The attackers' perspective is that users who receive a request will accept it, thus their personal information will be available to the owner of the cloned profile. The cloned applications presented in this paper bypass the privacy preferences of a user, and are able to harvest not only the user's information, but also the information of the user's friends.

On May 1, 2008, the BBC's technology program, analyzed in [20], how a special Facebook application they have created, could potentially harvest sensitive personal information from

Application	Installations	Accept	Deny	Deny & Accept
Lucky Art on Facebook	147	114 (77.5%)	42	9
Lucky Art on Friendster	136	66 (48.5%)	79	9
Lucky Art on Orkut	32	27 (84.3%)	7	2
How big is your brain?	404	264 (65.3%)	150	10
How smart are you?	175	114 (65.1%)	66	5
Are you happy right now?	61	37 (60.6%)	26	2
Are you normal or insane?	45	25 (55.5%)	20	0
Do you live your life to the fullest?	9	8 (88.8%)	1	0

TABLE I: The unique installations for the self-signed Java applets and the percentage of users who accepted or denied the digital signature presented to them.

users who installed it to their profile. It took them less than three hours, to create *Miner*, an evil data mining application. The malicious application was collecting users' personal details, and those of the users' friends, and was e-mailing them to the developers inbox.

Bonneau *et al.* in [27], listed several ways in which adversaries can extract users' personal details and social graph information from Facebook, on a large scale. Their harvesting methods include: (i) extract users' information from search engines, where Facebook exposes a limited public view of users' profile [26], (ii) creation of false fictitious Facebook profiles (iii) profile compromising, *e.g.* Bryan Rutberg's identity theft [21], and phishing, as the Facebook log-in page is not authenticated via TLS, (iv) malicious data mining applications and (v) exploiting security flaws in FQL queries.

Felt in [19], exploited the early design principles of the Facebook Platform, accomplishing to add arbitrary JavaScript to users' profiles. More precisely, this was mainly due to a Cross Site Scripting (XSS) vulnerability while parsing some attributes of the *<fb:swf/>* FBML tag. Shortly, after publication, Facebook fixed the vulnerability.

Authors in [38] exploited the friendship of an ordinary malicious user with a popular user (*e.g.* a famous celebrity or musician) who has a large friend circle, in order to cause a small-scale DDoS attack and create a botnet command and control channel. Their attack method benefitted from the ability to post HTML tags in comment boxes on users' profile pages on MySpace. Preparative to launch a DDoS attack they posted hot-links to large media files hosted by a victim web server. Thereby, the unsolicited HTTP GET requests coerced by hot-links, when someone visits the profile page of a popular user, could create a flash crowd.

VI. Conclusion

In this paper we presented techniques for building malicious applications in social networks and went over three proof-of-concept examples. An application that can launch DDoS attacks using a social network, one that can retrieve remote files from a user's machine and one that can leak user's private data. For all proof-of-concept applications we provided a brief evaluation. The main goal of this work is to highlight possible misuses of current technologies deployed in social networks.

References

[1] Facebook - Connect and share with the people in your life. http://www.facebook.com.

[2] Facebook API. http://wiki.developers.facebook.com/index.php/API.

[3] Facebook Developers. http://www.facebook.com/developers.

[4] Facebook JavaScript (FBJS). http://wiki.developers.facebook.com/index.php/FBJS.

[5] Facebook Markup Language (FBML). http://wiki.developers.facebook.com/index.php/FBML.

[6] Facebook Query Language (FQL). http://wiki.developers.facebook.com/index.php/FQL.

[7] Friendster. http://www.friendster.com.

[8] Friendster Developers Platform. http://www.friendster.com/developer/index.php?type=fbopen.

[9] Key and Certificate Management Tool. http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html.

[10] LinkedIn - Relationships Matter. http://www.linkedin.com.

[11] Lucky Art. http://www.facebook.com/apps/application.php?id=49562151481.

[12] MySpace. http://www.myspace.com.

[13] OpenSocial. http://www.opensocial.org.

[14] Orkut. http://www.orkut.com.

[15] Photo of the Day. http://www.facebook.com/apps/application.php?id=8752912084.

[16] XING - Social Network for Business Professionals. http://www.xing.com.

[17] MySpace XSS QuickTime Worm, 2006. http://securitylabs.websense.com/content/Alerts/1319.aspx.

[18] PC World: Hackers Crash the Social Networking Party, 2006. http://www.pcworld.com/article/127347.

[19] A. Felt. Defacing Facebook: A Security Case Study, 2007. http://www.cs.virginia.edu/felt/fbook/facebook-xss.pdf.

[20] BBC News: Identity 'at risk' on Facebook, 2008. http://news.bbc.co.uk/2/hi/programmes/click_online/7375772.stm.

[21] Facebook ID theft targets 'friends', 2009. http://redtape.msnbc.com/2009/01/post-1.html.

[22] Y.-Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. Analysis of Topological Characteristics of Huge Online Social Networking Services. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 835–844, New York, NY, USA, 2007. ACM.

[23] E. Athanasopoulos, A. Makridakis, S. Antonatos, D. Antoniades, S. Ioannidis, K. G. Anagnostakis, and E. P. Markatos. Antisocial Networks: Turning a Social Network into a Botnet. In *ISC '08: Proceedings of the 11th international conference on Information Security*, pages 146–160, Berlin, Heidelberg, 2008. Springer-Verlag.

[24] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group Formation in Large Social Networks: Membership, Growth, and Evolution. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 44–54, New York, NY, USA, 2006. ACM.

[25] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda. All Your Contacts Are Belong to Us: Automated Identity Theft Attacks on Social Networks. In *WWW '09: Proceedings of the 18th international conference on World Wide Web*, pages 551–560, New York, NY, USA, 2009. ACM.

[26] J. Bonneau, J. Anderson, R. Anderson, and F. Stajano. Eight Friends Are Enough: Social Graph Approximation via Public Listings. In *SNS '09: Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, pages 13–18, New York, NY, USA, 2009. ACM.

[27] J. Bonneau, J. Anderson, and G. Danezis. Prying Data out of a Social Network. In *Advances in Social Networks Analysis and Mining (ASONAM)*, July 2009.

[28] D. Boyd and N. B. Ellison. Social Network Sites: Definition, History, and Scholarship. *Journal of Computer-Mediated Communication*, 13(1), 2007.

[29] G. Brown, T. Howe, M. Ihbe, A. Prakash, and K. Borders. Social Networks and Context-Aware Spam. In *CSCW '08: Proceedings of the ACM 2008 conference on Computer supported cooperative work*, pages 403–412, New York, NY, USA, 2008. ACM.

[30] R. Dhamija, J. D. Tygar, and M. Hearst. Why Phishing Works. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 581–590, New York, NY, USA, 2006. ACM Press.

[31] A. Felt and D. Evans. Privacy Protection for Social Networking Platforms. *Web 2.0 Security and Privacy*, 2008.

[32] R. Gross, A. Acquisti, and H. J. Heinz, III. Information Revelation and Privacy in Online Social Networks. In *WPES '05: Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 71–80, New York, NY, USA, 2005. ACM.

[33] G. Hogben. Security Issues and Recommendations for Online Social Networks. Technical report, ENISA, October 2007.

[34] T. N. Jagatic, N. A. Johnson, M. Jakobsson, and F. Menczer. Social Phishing. *Commun. ACM*, 50(10):94–100, 2007.

[35] M. J. Kushin and K. Kitchener. Getting Political on Social Network Sites: Exploring Online Political Discourse on Facebook. In *Annual Convention of the Western States Communication Association*, Phoenix, AZ, 2009.

[36] V. T. Lam, S. Antonatos, P. Akritidis, and K. G. Anagnostakis. Puppetnets: Misusing Web Browsers as a Distributed Attack Infrastructure. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 221–234, New York, NY, USA, 2006. ACM.

[37] A. Mislove, M. Marcon, K. P. Gummadi, P. Drushcel, and B. Bhattacharjee. Measurement and Analysis of Online Social Networks. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 29–42, New York, NY, USA, 2007. ACM.

[38] B. E. Ur and V. Ganapathy. Evaluating Attack Amplification in Online Social Networks. In *W2SP '09: 2009 Web 2.0 Security and Privacy Workshop*, Oakland, California, May 2009.

[39] S. Utz. The (Potential) Benefits of Campaigning via Social Network Sites. *Journal of Computer-Mediated Communication*, 14(2):221–243, 2009.