Understanding the Behavior of Malicious Applications in Social Networks

Andreas Makridakis, Elias Athanasopoulos, Spiros Antonatos, Demetres Antoniades, Sotiris Ioannidis, and Evangelos P. Markatos,
Foundation for Research & Technology Hellas (FORTH)

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 article 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 to carry out network attacks. An adversary can take control of a visitor's session by remotely manipulating their browsers through legitimate web control functionality such as image-loading HTML tags, JavaScript instructions, and Java applets.

The popularity of online social networks (OSNs) [1] is ever increasing. The online communities created by OSNs are a fast growing phenomenon on the web empowered by new modes of social interaction among people from around the globe. OSNs are useful for keeping in touch with friends, forming new contacts, research collaboration, information sharing, political campaigns [2, 3], and so on. Some OSNs are used for professional contacts, such as LinkedIn [4] and XING [5], where a user can discover business connections, while others, such as Facebook [6], MySpace [7], and Orkut [8], are friendship-focused and primarily used for communication, photo and video sharing, and entertainment.

The massive adoption of OSNs by Internet users 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 distributed denial of service (DDoS) attacks, malware propagation, spamming, privacy violations, and disk compromise, to name a few. OSNs have some intrinsic properties that make them ideal for exploitation by an adversary:

- A very large and highly distributed user base
- Clusters of users sharing the same social interests, developing trust relationships, and seeking access to the same resources
- 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 article we explore these properties, develop real exploits, and analyze their impact.

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.

Platform Overview — Core Components

A user who wants to build a Facebook application must simply add the Developer Application [9] to her account. The application can be implemented in any development environment the user prefers. Facebook and third-party application developers have created a large number of client libraries in several development environments. 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 requirements:

- Provide it access to a Web server able to host the application.
- Upload the appropriate Facebook client library to the server.
- In case the application has any storage requirements (i.e., stores user information),[1] the user should also provide access to a relational database management system such as MySQL.

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

0890-8044/10/$25.00 © 2010 IEEE

• 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, and the IP address of the hosting web server. Some critical fields are:

• The Canvas page URL
• 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.

Typically, a few days after submitting the application the Facebook Platform Team notifies the developer that the application has been either successfully accepted or rejected. If the application is accepted, it is added to the Application Directory,[2] which allows users to easily find new applications and install them to their profile. Below we provide a short description of some basic components provided by Facebook for assisting application development.

Facebook Markup Language: The Facebook Markup Language (FBML) [10] 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 users and users' friends.

Facebook Query Language: The Facebook Query Language (FQL) [11] 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) [12] 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 sandboxing of the code. Furthermore, FBJS supplies powerful AJAX objects and an animation library to developers.

Facebook API: Using the Facebook application programming interface (API) [13], 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 a user's hometown location, high school information, favorite quotes, and so on. 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*

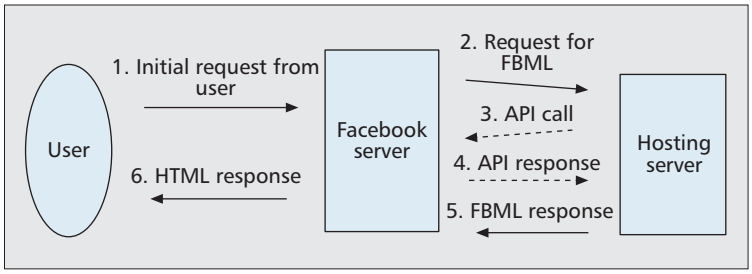

Figure 1. *How an FBML Canvas page is rendered.*

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 an application's contents to the user's browser. These steps are depicted in Fig. 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, 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 resulting 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.

Antisocial Activities

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

The OSN as a 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 (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 accesses the malicious application. We define as *FaceBots* the collection of Web browsers that are forced to generate requests upon running a malicious Facebook application (Fig. 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 toward the victim.

We created a real-world Facebook application, called *Photo of the Day* [14], which displays a different photo every day. When a Facebook user accesses the application, she 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 to installation. This practice helps further propagation of the application in Facebook.

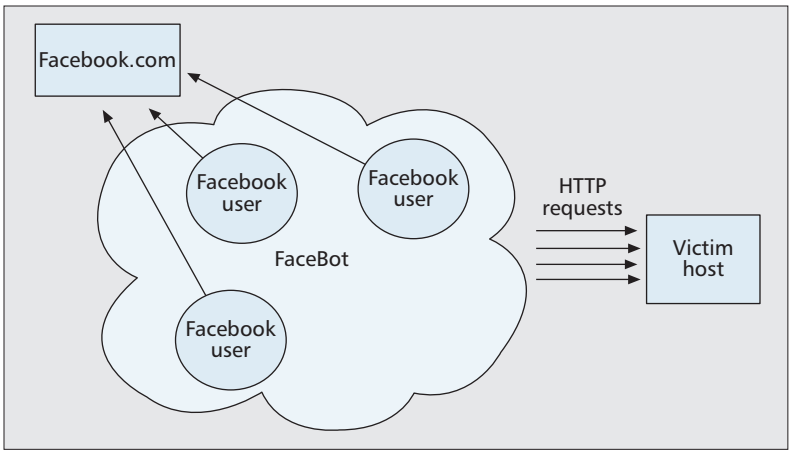

Figure 2. *The architecture of a FaceBot. Users access a malicious application on the social site (facebook.com), and subsequently a series of HTTP requests are created targeting the victim host.*

```
<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/>
```

Figure 3. *Sample code of a hidden frame inside a Facebook application that causes an image, image1.jpg, to be fetched from a remote host.*

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 Fig. 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 as `<fb:iframe/>` in FBML. The code listed in Fig. 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 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 serve 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 that do not utilize the above caching properties.

Hosting Issues — The adversary must also provide hosting for the malicious application, and be able to cope with requests from users accessing the application. However, this can be avoided using a free hosting service specifically designed for Facebook applications (e.g., 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.

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 her 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 profiles.

Users that access these applications have to upload a photo, which is transformed to colored HTML. Also, a random lucky color and number are provided; hence the term *Lucky*. Except for Lucky Art [15], 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 [16]. Below, we list the steps needed to create the self-signed file upload applet used by Lucky Art:

- keytool -genkey -keyalg rsa -alias FacebookLuckyArt
- keytool -export -alias FacebookLuckyArt -file FacebookLuckyArt.crt
- javac upload/FileUpload.java

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

[5] *The 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*

• jar cvfm applet.jar upload/*.class
• 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 per-
cent" height=107>
</applet>

</body>
</html>
```

We have also experimented with The Lucky Art application in two other online social networks, orkut.com [8] and friendster.com [17].

The developer platform of Orkut is based on OpenSocial [18], 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 the application. Thus, in order to build Lucky Art on Orkut, we simply provide an XML file.

Friendster has a Developer Platform, called fbOpen Platform [19], which 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 reject 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:*
• *Cannot install*
• *Got security warning when attempting to install*

Therefore, we removed the self-signed applet from Lucky Art and kept 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 the application's functionality to the original one, including the self-signed applet. Friendster never complained about the fundamental changes made to the application's functionality.

Personal Information Leakage

Facebook gives users the opportunity to have their profile locked and visible only to their friends. However, an attacker can collect sensitive personal information of Facebook users without their permission. A Facebook application has full access to the majority of a user's *and* the user's friends details by calling methods of the Facebook API.[8] Although users can set the privacy settings for each installed application, Felt *et al.* in [20] state that most users give all applications the right to have full access to their account details. An adversary could deploy an application that 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, and 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, 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 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.

The Facebook Platform imposes a timeout of nearly 10 s 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 Fig. 1, and concerning the round-trip time (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 to receive all the needed personal data. To overcome this, *iframes* can be used so that the basic HTML response is delivered in parallel to a Facebook server. We list below the FBML code included in the index page in order to embed a hidden frame responsible for fetching personal details of a 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>
```

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

[8] *For example, Users.getInfo: http://wiki.developers.facebook.com/index.php/Users.getInfo*

Application	Installations	Accept	Deny	Deny and 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 1. *The unique installations for the self-signed Java applets and the percentage of users who accepted or denied the digital signature presented to them.*

Experimental Evaluation

We now provide a brief evaluation of the proof-of-concept attacks based on self-signed Java applets outlined earlier. For the FaceBot attack a complete evaluation can be found in [21].

Self-Signed Applets Acceptance

For the self-signed Java applets that live on facebook.com, friendster.com, and orkut.com, we observe whether or not a user accepts the digital signature. In Table 1 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 1009 installations, where 655 users have accepted the requested digital signature. Therefore, it is possible to compromise 65 percent 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.

Related Work

Several researchers have conducted significant work on the structure and evolution of online social networks [22–24], 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 [25, 26], 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 [27]. Our work here extends the idea of Puppetnets by taking into account the characteristics of social networking web sites.

Jagatic *et al.* in [28], Hogben in [29], and Brown *et al.* in [30] study how phishing attacks [31] 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 simply sending mass emails to random people unrelated to each other. In [32] the authors found that a large amount of students at Carnegie Mellon University were 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 [33], 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, and their personal information will thus be available to the owner of the cloned profile. The cloned applications presented in this article 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 how a special Facebook application they have created could potentially harvest sensitive personal information from users who installed it to their profile [34]. It took them less than three hours to create Miner, an evil data mining application. The malicious application collected users' personal details and those of the users' friends, and emailed them to the developers' inbox.

Bonneau *et al.* listed several ways in which adversaries can extract users' personal details and social graph information from Facebook on a large scale [35]. Their harvesting methods include:

- Extract users' information from search engines, where Facebook exposes a limited public view of users' profile [36]
- Creation of fictitious Facebook profiles
- Profile compromising (e.g., Bryan Rutberg's identity theft [37] and phishing) as the Facebook log-in page is not authenticated via TLS
- Malicious data mining applications
- Exploiting security flaws in FQL queries

Felt exploited the early design principles of the Facebook Platform, accomplishing the addition of arbitrary JavaScript to users' profiles [38]. 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 [39] 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 benefited from the ability to post HTML tags in comment boxes on users' profile pages on MySpace. Preparatory to launching a DDoS attack they posted hot links to large media files hosted by a victim web server. Thus, when someone visits the profile page of a popular user, the unsolicited HTTP GET requests coerced by hot links could create a flash crowd.

Conclusion

In this article we present 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 machinek and one that can leak users' private data. For all proof-of-concept applications we provide a brief evaluation. The main goal of this work is to highlight possible misuses of current technologies deployed in social networks.

Acknowledgments

Andreas Makridakis, Elias Athanasopoulos, Spiros Antonatos, Demetres Antoniades, Sotiris Ioannidis, and Evangelos P. Markatos are also with the University of Crete. Elias Athanasopoulos is funded by the Microsoft Research Ph.D. Scholarship project, which is provided by Microsoft Research Cambridge.

References

[1] D. Boyd and N. B. Ellison, "Social Network Sites: Definition, History, and Scholarship," *J. Computer-Mediated Commun.*, vol. 13, no. 1, 2007.
[2] S. Utz, "The (Potential) Benefits of Campaigning via Social Network Sites," *J. Computer-Mediated Commun.*, vol. 14, no. 2, 2009, pp. 221–43.
[3] M. J. Kushin and K. Kitchener, "Getting Political on Social Network Sites: Exploring Online Political Discourse on Facebook," Annual Convention of the Western States Commun. Assn., Phoenix, AZ, 2009.
[4] LinkedIn — Relationships Matter, http://www.linkedin.com
[5] XING — Social Network for Business Professionals, http://www.xing.com
[6] Facebook — Connect and share with the people in your life; http://www.facebook.com
[7] MySpace, http://www.myspace.com
[8] Orkut, http://www.orkut.com
[9] Facebook Developers, http://www.facebook.com/developers
[10] Facebook Markup Language (FBML), http://wiki.developers.facebook.com/index.php/FBML
[11] Facebook Query Language (FQL), http://wiki.developers.facebook.com/index.php/FQL
[12] Facebook JavaScript (FBJS), http://wiki.developers.facebook.com/index.php/FBJS
[13] Facebook API, http://wiki.developers.facebook.com/index.php/API
[14] Photo of the Day, http://www.facebook.com/apps/application.php?id=8752912084
[15] Lucky Art, http://www.facebook.com/apps/application.php?id=49562151481
[16] Key and Certificate Management Tool, http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html
[17] Friendster, http://www.friendster.com
[18] OpenSocial, http://www.opensocial.org
[19] Friendster Developers Platform, http://www.friendster.com/developer/index.php?type=fbopen
[20] A. Felt and D. Evans, "Privacy Protection for Social Networking Platforms," *Web 2.0 Security and Privacy*, 2008.
[21] E. Athanasopoulos *et al.*, "Antisocial Networks: Turning a Social Network into a Botnet," *ISC '08, Proc. 11th Int'l. Conf. Info. Sec.*, Berlin, Heidelberg, 2008, Springer-Verlag, pp. 146–60.
[22] A. Mislove *et al.*, "Measurement and Analysis of Online Social Networks," *IMC'07, Proc. 7th ACM SIGCOMM Conf. Internet Measurement*, New York, NY, 2007, pp. 29–42.
[23] Y.-Y. Ahn *et al.*, "Analysis of Topological Characteristics of Huge Online Social Networking Services," *WWW '07: Proc. 16th Int'l. Conf. World Wide Web*, New York, NY, 2007, ACM, pp. 835–44.
[24] L. Backstrom *et al.*, "Group Formation in Large Social Networks: Membership, Growth, and Evolution," *KDD '06, Proc.12th ACM SIGKDD Int'l. Conf. Knowledge Discovery and Data Mining*, New York, NY, 2006, pp 44–54.
[25] World: Hackers Crash the Social Networking Party, 2006, http://www.pcworld.com/article/127347
[26] MySpace XSS QuickTime Worm, 2006, http://securitylabs.websense.com/content/Alerts/1319.aspx
[27] V. T. Lam *et al.*, "Puppetnets: Misusing Web Browsers as a Distributed Attack Infrastructure," *CCS'06: Proc. 13th ACM Conf. Comp. and Commun. Sec.*, New York, NY, 2006, pp. 221–34.
[28] T. N. Jagatic *et al.*, "Social Phishing," *Commun. ACM*, vol. 50, no. 10, 2007, pp. 94–100.
[29] G. Hogben, "Security Issues and Recommendations for Online Social Networks," tech. rep., ENISA, Oct. 2007.
[30] G. Brown *et al.*, "Social Networks and Context-Aware Spam," *CSCW '08: Proc. ACM 2008 Conf. Comp. Supported Cooperative Work*, New York, NY, 2008, pp. 403–12.
[31] R. Dhamija, J. D. Tygar, and M. Hearst, "Why Phishing Works," *CHI '06, Proc. SIGCHI Conf. Human Factors in Computing Dystems*, New York, NY, 2006, pp. 581–90.
[32] R. Gross, A. Acquisti, and H. J. Heinz, III, "Information Revelation and Privacy in Online Social Networks," *WPES'05: Proc. 2005 ACM Wksp. Privacy in the Electronic Society*, New York, NY, 2005, pp. 71–80.
[33] L. Bilge *et al.*, "All Your Contacts Are Belong to Us: Automated Identity Theft Attacks on Social Networks," *WWW '09, Proc. 18th Int'l. Conf. World Wide Web*, New York, NY, 2009, pp. 551–60.
[34] BBC News: Identity 'at risk' on Facebook, 2008; http://news.bbc.co.uk/2/hi/programmes/click_online/7375772.stm
[35] J. Bonneau, J. Anderson, and G. Danezis, "Prying Data out of a Social Network," *Advances in Social Networks Analysis and Mining*, July 2009.
[36] J. Bonneau *et al.*, "Eight Friends Are Enough: Social Graph Approximation via Public Listings," *SNS'09: Proc. 2nd EuroSys Wksp. Social Network Sys.*, New York, NY, 2009, pp. 13–18.
[37] Facebook ID theft targets 'friends,' 2009, http://redtape.msnbc.com/2009/01/post-1.html
[38] A. Felt, Defacing Facebook: A Security Case Study, 2007, http://www.cs.virginia.edu/felt/fbook/facebook-xss.pdf
[39] B. E. Ur and V. Ganapathy, "Evaluating Attack Amplification in Online Social Networks," *W2SP '09: 2009 Web 2.0 Security and Privacy Wksp.*, Oakland, CA, May 2009.

Biographies

Andreas Makridakis (amakrid@ics.forth.gr) was a research assistant in the Distributed Computing Systems laboratory division at the Institute of Computer Science of the Foundation for Research and Technology — Hellas (FORTH-ICS) from 2006 to 2009. He received his B.Sc. degree in computer science from the University of Crete. He also holds an M.Sc. degree from the University of Crete in association with FORTH-ICS for his study on "Networks: Turning a Social Network into an Attack Platform." Most of his research centers around passive network monitoring and security in distributed computing systems, focusing mainly on online social networks.

Elias Athanasopoulos (elathan@ics.forth.gr) received a B.Sc. in physics from the University of Athens and an M.Sc. in computer science from the University of Crete in 2004 and 2006, respectively. He is currently a Ph.D. candidate in computer science at the University of Crete. He has published at ACNS, IWSEC, ISC, W2SP, USENIX WebApps, and ESORICS. He is a research assistant with FORTH and has received a Ph.D. scholarship from Microsoft Research (Cambridge) for the period 2008–2011.

Spiros Antonatos (antonat@ics.forth.gr) received his diploma in computer science from the University of Crete in 2003, and M.Sc. and Ph.D. degrees from the same department in 2005 and 2009, respectively. Since 2002 he is a research assistant at the DCS laboratory at FORTH-ICS. His main research interests are honeypot technologies, web security, anonymization techniques, and network intrusion detection systems. He has been the technical manager of the NoAH project, funded in part by the European Union, aimed at the design and implementation of collaborative honeypot infrastructures. He has also participated in the EU-funded projects SCAMPI (network monitoring), LOBSTER (network monitoring), FORWARD (future and emerging threats), and WOMBAT (malware and threat observatory).

Demetres Antoniades (danton@ics.forth.gr) received his M.Sc. and B.Sc. degrees in computer science from the University of Crete, in 2005 and 2007 respectively. He is currently a Ph.D. candidate in Computer Science in the same university. Since 2004, he is also with the FORTH-ICS, Crete, where he currently works in the Distributed Computing Systems Lab. His main research interests include network monitoring and traffic classification.

Sotiris Ioannidis (sotiris@ics.forth.gr) received a B.Sc. degree in mathematics and an M.Sc. degree in computer science from the University of Crete in 1994 and 1996, respectively. In 1998 he received an M.Sc. degree in computer science from the University of Rochester, and in 2005 he received his Ph.D. from the University of Pennsylvania. He held a research scholar position at Stevens Institute of Technology until 2007; since then he has been an associate researcher at FORTH-ICS) and an adjunct professor at the Computer Science Department of the University of Crete. His research interests are in the area of systems and network security, security policy, privacy, and high-speed networks. He has chaired and served on numerous program committees of prestigious conferences. He is a Marie-Curie Fellow and is currently running the PASS European project.

Evangelos P. Markatos (markatos@ics.forth.gr) received his diploma in computer engineering from the University of Patras in 1988, and M.S. and Ph.D. degrees in computer science from the University of Rochester, New York, in 1990 and 1993, respectively. Since 1992 he has been an associate researcher at FORTH-ICS, where he is currently head of the Distributed Computing Systems Laboratory. Since 1994 he has also been with the Computer Science Department at the University of Crete, where he is currently a full professor. He conducts research in the areas of distributed systems, Internet systems and technologies, as well as computer and communication systems security. He is the author of more than 100 papers and book chapters. He is currently the coordinator of research projects funded by the European Union, by the Greek government, and by private organizations.