Antisocial Networks: Turning a Social Network into a Botnet

E. Athanasopoulos[1], A. Makridakis[1], S. Antonatos[1], D. Antoniades[1], S. Ioannidis[1], K. G. Anagnostakis[2], E. P. Markatos[1]

[1] Institute of Computer Science (ICS)
Foundation for Research & Technology Hellas (FORTH)
{elathan,amakrid,antonat,danton,sotiris,markatos}@ics.forth.gr
[2] Institute for Infocomm Research, Singapore
kostas@i2r.a-star.edu.sg

Abstract. *Antisocial Networks* are distributed systems based on social networking Web sites that can be exploited by attackers, and directed to carry out network attacks. Malicious users are able to take control of the visitors of social sites by remotely manipulating their browsers through legitimate Web control functionality such as image-loading HTML tags, JavaScript instructions, *etc.* In this paper we experimentally show that Social Network web sites have the ideal properties to become attack platforms.

We start by identifying all the properties of Facebook, a real-world Social Network, and then study how we can utilize these properties and transform it into an attack platform against any host connected to the Internet. Towards this end, we developed a real-world Facebook application that can perform malicious actions covertly. We experimentally measured it's impact by studying how innocent Facebook users can be manipulated into carrying out a Denial-of-Service attack. Finally, we explored other possible misuses of Facebook and how they can be applied to other online Social Network web sites.

1 Introduction

The massive adoption of social networks by Internet users provides us with a unique opportunity to study possible exploits that will turn them into platforms for antisocial and illegal activities, like DDoS attacks, malware propagation, spamming, privacy violations, *etc.* We define *antisocial networks* as a *social network, deviously manipulated for launching activities connected with fraud and cyber-crime.*

Social networks have by nature some intrinsic properties that make them ideal to be exploited by an adversary. The most important of these properties are: (*i*) a very large and highly distributed user-base, (*ii*) clusters of users sharing the same social interests, developing trust with each other, and seeking access to the same resources, and (*iii*) platform openness for deploying fraud resources and applications that lure users to install them. All these characteristics give adversaries the opportunity to manipulate massive crowds of Internet users and

force them to commit antisocial acts against the rest of the Internet, without their knowledge. In this paper we explore these properties, develop a real exploit, and analyze its impact.

The main contributions of this paper is a first investigation into the potential misuse of a social network for launching DDoS attacks on third parties. We have built an actual Facebook application, that can turn its users into a FaceBot. We used our FaceBot to carry out a complete evaluation of our proof-of-concept attack via real-world experiments. Extrapolating from these measurements along with popularity metrics of current Facebook applications, we show that owners of popular Facebook applications have a highly distributed platform with significant attack firepower under their control.

2 Related Work

The structure and evolution of social networks has been extensively studied [18, 9, 11], but little work has been done on measuring real attacks on these sites. The most closely related work to our paper was done by Lam *et al.* in [17]. Our work here extends the idea of Puppetnets by taking into account the characteristics of a special kind of Internet systems which rely heavily on the social factor: social network web sites. The authors of [17] omit explaining *how* they will make their Web site popular, in order to carry out the attack. We on the other hand are taking advantage of already popular Web sites like `facebook.com`. Such sites prove to be ideal for carrying Puppetnet type attacks.

Jagatic *et al.* in [16] study how phishing attacks [13] can be made more powerful by extracting 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. However, apart from scattered blog entries that report isolated attacks (such as malware hosting in Myspace [4]), there have been no large-scale attacks to social networks, or using social networking sites, reported or studied so far.

In the space of peer-to-peer systems, there have been a few attacks that have appeared and have been analyzed by researchers. One may consider a peer-to-peer system to be similar to a social network in the sense that there are millions of users that connect to each other forming a network. Gnutella, an unstructured peer-to-peer file sharing system, has been used in the past as an attack platform [10]. In a similar fashion, the work in [19, 21] presented how Overnet and KAD can be misused for launching Denial of Service attacks to third parties. Finally, in [12], the authors have managed to transform BitTorrent to a platform for similar attacks.

3 Background

Social Networks. Social networking sites are becoming more popular by the day. Millions of people daily use social networking sites such as `facebook.com`,

`LinkedIn.com`, `Myspace.com` and `Orkut.com`. Some of them are used for professional contacts, *e.g.* LinkedIn, while others are primarily used for communication and entertainment. The structure of a social networking site is quite simple. Users register to the site, create their profile describing their interests and putting some personal information, and finally add friends/contacts to their profile. Adding a friend involves a confirmation step from the other party most of the times. The view of a user's profile is usually limited to the friends of that user, unless the user wants the profile to be public. In that case, all users of the site can view it. Social networking sites also support the creation of groups and networks.

Facebook is considered to be one of the most popular social networking sites. It started as a project of a student to keep track of schoolmates but has now grown up to serve more than 64 million people from around the world, with an average of 250,000 new registrations per day [2]. Facebook has a very interesting feature, the Facebook applications. Facebook builders have implemented a platform on top of which developers can build complete applications. In the *Facebook Platform* any developer with a good idea and basic programming skills can create one. Over 200,000 developers have done so, as reported by Adonomics [1]. Users can add these applications to their profile and invite their friends to add them too. A constraint put by Facebook is that invitations are limited to up to 20 friends per day. Typical applications involve solving a quiz, filling questionnaires, playing games and many more. Up to date, the number of Facebook applications has surpassed fifteen thousand. Facebook applications can be considered as XHTML snippets that inherit all properties of web applications.

Puppetnets. Puppetnets [17] exploit the design principles of World Wide Web. Web pages can include links to elements located at different domains, other than the one they are hosted at. A malicious user can craft special pages that contain thousands of links pointing at a victim site. When an unsuspecting user visits that page, her browser starts downloading elements from the victim site and thus consuming its bandwidth. The firepower of this attack increases with the popularity of the malicious page, similar to the slashdot effect [15].

Puppetnets use a number of techniques to make the attacks more effective. The use of JavaScript permits more flexible and powerful attacks as unsuspecting users can repeatedly download elements from victim sites or perform other kinds of attacks, such as port scanning and computational attacks. The firepower of Puppetnets depends on three main factors. First, the popularity of the malicious page. Second, the duration of visits to the malicious page. The more the unsuspecting user stays on the malicious page, the longer the attack takes place in the background. Third, the bandwidth of unsuspecting users and their latency to the victim site. These factors determine the number of downloads per second an attacker can achieve.

4 Experimental Evaluation

In this section we experimentally evaluate the firepower of a FaceBot. Specifically, we explore the effect of placing a malicious Facebook application, which exports HTTP requests to a victim host. We have conducted experiments, using a *least effort* approach. By using the term of *least effort* we mean that during the whole study we did the *least* we could do in terms of spending resources, adding complexity and enhancing our developments with obscure and hackish features, which could lead in overestimated results. For example, during the deployment of a Facebook application we *did not add special obligatory massive invitation features* for boosting the application's propagation in the social network. In section 5, based on our experimental results, we extrapolate the firepower of FaceBot, by examining the popularity of existing Facebook applications.

4.1 Experimental Setup

Our initial vision is to create a first *proof-of-concept* FaceBot for demonstration purposes, while at the same time not causing any harm to real Facebook users. Furthermore, our experiment was conducted using the *real* social network website, namely `facebook.com`.

We created a real-world Facebook application, which we call *Photo of the Day* [8], that presents a different photo from National Geographic to facebook users every day. In order to keep the experiment in a *least effort* approach, we didn't employ any obligatory invitations during its installation in a user's profile.[3] However, we did announce the application to members of our research group and we encouraged them to propagate the application to their colleagues. To our surprise, the application was installed by a significant Facebook population, which was completely unaware to us (see our popularity results, later in this section).

Every time a user clicks on the *Photo of the Day* application, an image from the respective service of National Geographic[4] appears [7]. However, we have placed special code in the application's source code, so that every time a user views the photo, HTTP requests are generated towards a victim host. More precisely, the application embeds four hidden frames with inline images hosted at the victim. Each time the user clicks inside the application, the inline images are fetched from the victim, causing the victim to serve a request of 600 KBytes, but the user is not aware of that fact (the images are never displayed). We list a portion of our sample source code which is responsible for fetching an inline

[3] It is very common that Facebook applications require a user to invite a subset of her friends, and thus advertize the application to the Facebook community, prior the installation. This practice helps in the further propagation of the application in Facebook. Typically, a user must announce the application to about 20 of her friends in order to proceed with the installation.

[4] National Geographic has specific terms for content distribution, which are not violated by this work[6].

```
<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. 1. Sample code of a hidden frame, inside a Facebook application, which causes an image, namely `image1.jpg` to be fetched from `victim-host`.

image from a victim host and placing it to a hidden frame inside the *Photo of the Day* application, in Figure 1.

For our experiments, the victim Web server which hosts the inline images is located in our lab, isolated from any other network activity. In the following subsection we present the results associated with the traffic experienced by our Web server.

4.2 Attack Magnitude

In Figure 2 we present the number of requests per hour recorded by our Web server from the time the *Photo of the Day* application was uploaded to `facebook.com` and for a period of a few days. Notice, that the request rate reached a peak of more than 300 requests/hour after a few days from the publication time. During the peak day of January 29th, our Web server recorded an excess of 6 Mbit per second of traffic (see Figure 3). The request rate shown in Figure 2, as well as the outgoing traffic shown in Figure 3, is purely Facebook related. We can isolate the packets originating from users accessing `facebook.com` by inspecting the *referer* field[5]. We further discuss the importance of the referer field in Section 6.

It is important to note that the request rate per hour never fell below a few tens of request and during peak times it reached a few hundred of requests. Notice, that depending on the nature of the malicious Facebook application, the request rate may differ substantially. In our experiment, each user was generating only four requests towards our Web server per application visit. We further explore the nature of a malicious Facebook application in Section 5.

It is also interesting to notice that the traffic pattern is quite bursty (see Figure 3). This is related to the *social nature* of the attack platform. Users seem to visit Facebook also in bursty fashion (approximately at the same time). This is more clearly presented in Figure 4, where we plot the distribution of user inter-arrival times (the times at which users visit the Photo of the Day application) for

[5] `http://www.w3.org/Protocols/HTTP/HTRQ_Headers.html#z14`

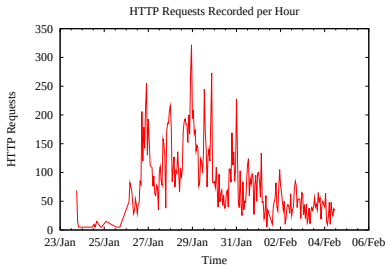

Fig. 2. The HTTP requests as were recorded by the victim Web server.

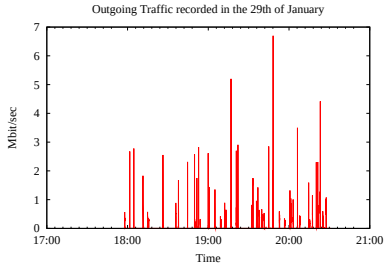

Fig. 3. Bandwidth use at the victim Web server during the attack on 29/01/2008.

the 29th of January. We calculated this distribution using the entry points to the Photo of the Day application as they were recorded by our victim Web server. The users' inter-arrival distribution indicates that a typical inter-arrival time has a period from a few tens of seconds to a few minutes. Note, that during the 29th of January, according to Figure 8, our proof of concept application recorded 480 Facebook daily active users.

To further verify our feelings about the bursty nature of the traffic we were experiencing in the victim host, we installed two sensors and captured traffic emitted by Facebook users. The first sensor was installed in an academic institute and was able to monitor approximately 120,000 IP addresses. We recorded 100 unique Facebook users in a monitoring period of 1 day. The second sensor was installed in a /16 enterprise network. We recorded 75 unique Facebook users in a monitoring period of 5 days. We used the collected traces from these sensors in order to calculate the user requests' inter-arrival distribution at Facebook. We present the results in Figure 5. It is evident that small inter-arrival periods characterize the requests made by Facebook users. Note, that users arrive in bursts to their home pages in facebook.com, but this does not immediately imply that they will use the Photo of the Day application.

To summarize, based on the spontaneous peaks in Figures 2 and 3, and considering the fact that Facebook users are arriving nearly at the same time (see Figure 4), we conclude that a malicious Facebook application can absorb Facebook users and force them to generate HTTP requests to a victim host in burst mode fashion.

Notice, that our malicious application was absorbing a fixed amount of traffic from the victim host. An adversary could employ more sophisticated techniques and create a JavaScript snippet, which continuously requests documents from a victim host over time. In this way the attack may be significantly amplified. In Figure 6 we plot typical session times of Facebook users, as were recorded by our two sensors. Observe that a typical user session of a Facebook user ranges from a few to tens of minutes.

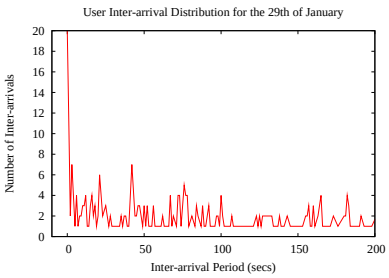

Fig. 4. The distribution of user inter-arrival times at the victim site on 29/01/2008, with over 480 users recorded as active.

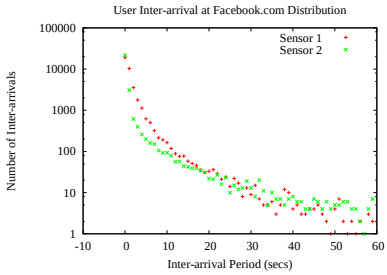

Fig. 5. The distribution of user inter-arrival periods at `facebook.com` for one day. Our two sensors recorded 100 and 75 unique users respectively.

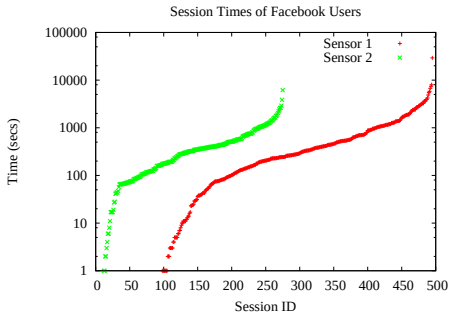

Fig. 6. Session times of Facebook users as were recorded by our two sensors. The first sensor recorded 495 user sessions and the other one recorded 275 user sessions.

4.3 Attack Distribution

Using the IP addresses recorded in the logs of our victim Web server, we tried to identify the geographical origin of each Facebook user. Our main interest was to investigate how distributed can an attack based on a social web site, like `facebook.com`, be. We used the `geoip` tool[3], in order to map our collected IPs to actual countries. We ignored the fact that some Facebook users might be using some sort of an anonymizing system like TOR [14], because our goal was not to capture the *origin of the users*, but the *origin of the requests*, which were recorded by our victim host.

In Figure 7 we are marking in black every country from which we recorded at least one request. It is evident that the nature of a FaceBot, even one that is a proof of concept, is highly distributed.

4.4 Tracking Popularity

In Figure 8 we explore the popularity of our proof of concept Facebook application, as it is measured by Adonomics [1]. Recall that, as we stated multiple

Fig. 7. Location of FaceBot hosts. Countries coloured in black hosted at least one FaceBot participant.

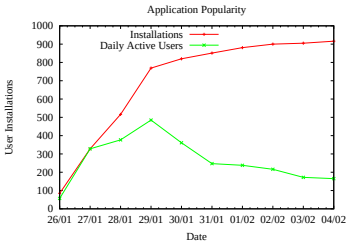

Fig. 8. The popularity of the *Photo of the Day* application, as it is tracked by Adonomics.com.

times in this section, we followed a *least effort* approach, which means that we did not employed sophisticated methods for advertizing our application to `facebook.com`. However, as it is evident from Figure 8, our application was installed by nearly 1,000 different users in the first few days. This is rather impressive correlating it with statistics related to commodity software downloads. For example, it took months for the most successful project in SourceForge.com to reach thousands of downloads[6].

5 Attack Firepower

Based on the experimental results from the previous section we proceed to estimate the firepower of a large FaceBot. For this we are going to assume that an adversary has developed a *highly popular* Facebook application, which employs the tricks we presented in the previous sections.

We denote with $F(t)$ the distribution of outgoing traffic a victim Web server exports, due to Facebook requests, over time. This is essentially the firepower of a FaceBot. In section 4 we experimentally measured this distribution for a proof of concept FaceBot and we presented our results in Figure 3. Our aim, in this section, is to find an analytical expression for $F(t)$.

We denote with a_{out} the outgoing traffic a Facebook application can pull from a victim host, once the user on that host is tricked into using the malicious application. Even though sophisticated use of client side technologies (like JavaScript) can make a_{out} a function over time (*e.g.*, a malicious JavaScript snippet can generate requests towards a victim host in an infinite loop), for simplicity we assume that a_{out} is a fixed quantity.

We denote with $U(t)$ the number of users accessing this application over time. It follows that:

$$F(t) = a_{out}U(t) \tag{1}$$

[6] eMule Statistics: `http://sourceforge.net/project/stats/?group_id=53489&ugn=emule&type=&mode=alltime`

To estimate $U(t)$, we need the following: (a) the number of active users over a period P, and (b) an estimation of the users' inter-arrival times. If we denote the active users with $u(t)$ and the inter-arrival distribution with $u_r(t)$, then:

$$U(t) = \frac{\int_0^P u(t)dt}{u_r(t)} \tag{2}$$

Assuming that there is a FaceBot based on a highly popular Facebook application and that we want to estimate its firepower at time T, F_T, we can use the average of the inter-arrival distribution, and thus:

$$F_T = a_{out}\frac{\int_0^P u(t)dt}{< u_r >} \tag{3}$$

For example, if we have a FaceBot with $a_{out} = 10Kbit/sec$, which is installed by 1,000 users, from whom 100 were active in the period of 10 seconds and their average inter-arrival time was 2 secs, then $F_{(10)} = 10Kbit/sec\frac{100}{2} = 0.5Mbit/sec$.

In Table 1 we list the Top-5 Facebook applications as of early February 2008, according to Adonomics.com[1]. These applications have from 1 million to more than 2 millions of daily active users. The user-base of these applications is so large, that we can assume that the user inter-arrival time follows a uniform distribution.[7] We further assume that an adversary has deployed one of these applications, which has 2 million of daily active users. That is, assuming uniform user inter-arrival time, approximately 23 users/sec are using the application. If the adversary has deployed the malicious application with $a_{out} = 1Mbit/sec$[8], then the victim will have to cope with unsolicited traffic of 23 Mbit/sec and during the period of one day will have received nearly 248 GB of unwanted data.

Application	Installations	Daily Active Users
FunWall	23,797,800	2,379,780
Top Friends	24,955,200	2,245,970
Super Wall	23,274,800	1,861,980
Movies	15,934,700	1,274,780
Bumper Sticker	7,989,700	1,118,560

Table 1. The Top-5 of Facebook applications as of the beginning of February 2008, in terms of active users. Source: Adonomics.com[1].

[7] Having a non-uniform inter-arrival time distribution would further amplify the attack, because the victim host would have to cope with large flash crowd events [15] in very short periods.

[8] The adversary needs to download a file of size of 125 KBytes from the victim, in order to achieve such an a_{out} value.

6 Discussion and Countermeasures

From our analysis in Section 5 we can see that an adversary can take full advantage of popular social utilities, to emit a high amounts of traffic towards a victim host. However, apart from launching a DDoS attack to third parties, there are other possible misuses in the fashion of Puppetnets [17]:

- *Host Scanning:* Using JavaScript, an attacker can make an application that identifies whether a host has arbitrary ports open. As browsers impose only few restriction on destination ports (some browsers like Safari even allow connection to sensitive ports like 25), an attacker can randomly select a host and a port, and request an object through normal HTTP requests. Based on the response time, which can be measured through Javascript, the attacker can figure if the port is alive or not.
- *Malware Propagation:* An unsuspecting user can participate in malware and attack propagation. If a server can be exploited by a URL-embedded attack vector, then malicious facebook applications can contain this exploit. Every user that interacts with the application will propagate the attack vector.
- *Attacking Cookie-based Mechanisms:* Similarly to XSS worms, a malicious application can override authentication mechanisms that are based on cookies. Badly-designed sites that support automated login using cookies suffer from such attacks.

Finally, there are other possible misuses of `facebook.com` itself. For example, an adversary can collect sensitive information of `facebook.com` users, without their permission. Facebook.com gives users the opportunity to have their profile locked and visible only by their contacts. However, a `facebook.com` application has full access in all user's details. An adversary could deploy an application, which simply posts all 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.[9]

In the rest of this section we propose countermeasures for defending and preventing a FaceBot based attack.

6.1 Defending against a FaceBot

To defend against a FaceBot, a victim host must filter out all incoming traffic introduced by Facebook users. Using the referer field of the HTTP requests the victim can determine whether a request originates from `facebook.com` or not, and stop the attack traffic (*e.g.* by using a NIDS or Firewall system). However, it is possible for a Facebook application developer to overcome this situation. With respect to our proof of concept application, which embeds hidden frames with inline images, the strategy would be to create a separate page to load them from. For example the source of the inline frame can be:

[9] Indeed, this proved to be possible, while this paper was under the review process[5].

```
src="http://attack-host/dummy-page?ref=victim-host/image1.jpg"
```

In this example the *attack host* is the Web server where the source code of the *Photo of the Day* lives. The dummy-page PHP file contains the following code:

```
<?php
if ($_GET["ref"]) { $ref=$_GET["ref"]; }
print("<meta http-equiv='refresh'
content='0; url=$ref'>");
?>
```

By employing this technique, HTTP requests received by the victim host have an empty referer field, giving the attacker a way to hide her identity. This is a typical usage of a reflector [20] by the adversary. Notice however, that the adversary must tunnel the requests to the victim. This means, that the adversary will also receive all the requests targeting the victim, but she will not have to *actually serve* the requests. Practically, the adversary will receive plain HTTP requests (a few bytes of size each), will have to process them in order to trim the referer related data and then pass it to the victim. On the other hand, the victim will have to serve the requests, which, depending on the files the victim serves, might reach the size of MBytes of information for each server request.

6.2 Preventing a FaceBot

Providers of social networks should be careful when designing their platform and APIs in order to have low interactions between the social utilities they operate and the rest of the Internet. More precisely, social network providers should be careful with the use of client side technologies, like JavaScript, *etc.* A social network operator should provide developers with a strict API, which is capable of giving access to resources only related to the system. Also, every application should run in an isolated environment imposing constraints to prevent the application from interacting with other Internet hosts, which are not participants of the social network. Finally, operators of social networks should invest resources in verifying the applications they host. Regarding our application, the Facebook Platform can cancel the use of `fb:iframe` tag, as this tag is used to load images hosted at the victim host. Currently, developers can not use `fb:iframe` tag on the profile page of a user. [10] Otherwise, the `fb:iframe` tag can be handled in a special manner, as in the case of the `img` tag. When publishing a page, Facebook servers request any image URL and then serve these images, rewriting the `src` attribute of all `img` tags using a *.facebook.com domain. This protects the privacy of Facebook's users and not allow malicious applications to extract information from image requests made directly from a the view of a user's browser. Thus, if the `src` attribute of an `iframe` is an image file (*e.g.* .jpg, .png, *etc.*), the Facebook Platform can handle these frames in a way similar to `img` tags.

[10] `http://wiki.developers.facebook.com/index.php/Fb:iframe`

7 Conclusion

In this paper we presented *Antisocial Networks* or how it is possible to turn a social network into a botnet that can be used to carry out a number of attacks. We developed FaceBot, an application that can run on `facebook.com`, and carry out DDoS attacks against any host on the internet. Our analysis involved building a real-world facebook.com application, conducting an actual attack on our lab servers, and doing an estimation of the firepower of a FaceBot.

We have shown that applications that live inside a social network can easily and very quickly attract a large user-base (in the order of millions of users) that can be redirected to attack a victim host. We experimentally determined the user-base to be highly distributed, and of a world-wide scale. Finally, we have shown that the victim of a FaceBot attack may be subject to an attack that will cause it to serve data of the magnitude of GigaBytes per day.

Acknowledgments

This work was supported in part by the project CyberScope, funded by the Greek Secretariat for Research and Technology under contract number PENED 03ED440. The work was, also, supported by the Marie Curie Actions - Reintegration Grants project PASS. We thank the anonymous reviewers for their valuable comments. Elias Athanasopoulos, Andreas Makridakis, Sotiris Ioannidis, Spiros Antonatos, Demetres Antoniades and Evangelos P. Markatos are also with the University of Crete. Elias Athanasopoulos is also funded from the PhD Scholarship Program of Microsoft Research Cambridge.

References

1. Facebook Analytics and Advertising. `http://adonomics.com`.
2. Facebook Statistics. `http://www.facebook.com/press/info.php?statistics`.
3. Geo IP Tool. `http://www.geoiptool.com`.
4. Hackers crash the Social Networking Party. `http://www.pcworld.com/article/id,127347-page,1-c,internettips/article.html`.
5. Identity 'at risk' on Facebook. `http://news.bbc.co.uk/2/hi/programmes/click_online/7375772.stm`.
6. National Geographic Content Usage. `http://www.nationalgeographic.com/community/terms.html#content`.
7. National Geographic Photo of the Day Utility. `http://photography.nationalgeographic.com/photography/photo-of-the-day`.
8. Photo of the Day. `http://www.facebook.com/apps/application.php?id=8752912084`.
9. Y.-Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. Analysis of Topological Characteristics of Huge Online Social Networking Sites. In *Proceedings of the 16th International Conference on World Wide Web*, May 2007.

10. E. Athanasopoulos, K. G. Anagnostakis, and E. P. Markatos. Misusing Unstructured P2P Systems to Perform DoS Attacks: The Network That Never Forgets. In J. Zhou, M. Yung, and F. Bao, editors, *ACNS*, volume 3989 of *Lecture Notes in Computer Science*, pages 130–145, 2006.
11. L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group Formation in Large Social Networks: Membership, Growth, and Evolution. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD06)*, August 2006.
12. K. E. Defrawy, M. Gjoka, and A. Markopoulou. Bottorrent: Misusing bittorrent to launch ddos attacks. In *Proceedings of the USENIX 3rd Workshop on Steps Towards Reducing Unwanted Traffic on the Internet (SRUTI)*, 2007.
13. R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in com puting systems*, pages 581–590, New York, NY, USA, 2006. ACM Press.
14. R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.
15. Halavais, A. The Slashdot Effect: Analysis of a Large-Scale Public Conversation on the World Wide Web. 2001.
16. T. N. Jagatic, N. A. Johnson, M. Jakobsson, and F. Menczer. Social phishing. *Commun. ACM*, 50(10):94–100, 2007.
17. 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.
18. A. Mislove, M. Marcon, K. P. Gummadi, P. Drushcel, and B. Bhattacharjee. Measurement and Analysis of Online Social Networks. In *Proceedings of the Internet Measurements Conference (IMC) 2007*, 2007.
19. N. Naoumov and K. Ross. Exploiting P2P systems for DDoS attacks. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, page 47, New York, NY, USA, 2006. ACM Press.
20. V. Paxson. An analysis of using reflectors for distributed denial-of-service attacks. *SIGCOMM Comput. Commun. Rev.*, 31(3):38–47, 2001.
21. M. Steiner, E. W. Biersack, and T. En-Najjary. Exploiting kad: Possible uses and misuses. *Computer Communication Review, 37(5)*, 2007.

Appendix

Facebook Architecture

Facebook provides all the essentials needed for easy deployment of applications that live inside the social network itself. A user who wants to build a Facebook application must simply add the *Developer Application*[11] to her account. The server side part of the application can be developed in PHP or Java. One major requirement is the presence of a Web server for hosting the new application. Using the Developer Application the developer fills out a form and submits the application. The form has fields, such as the application's name, the IP address of the Web server, *etc.* Typically, after a few days the Facebook Platform Team notifies the developer either that the application was successfully accepted or that it was rejected. Facebook Platform provides the Facebook Markup Language[12] (FBML), which is a subset of HTML along with some additional tags specific to Facebook. Also, the Facebook Query Language[13] (FQL) allows the developer to use an SQL-style interface to easily query some Facebook social data, such as the name or profile picture of a user. Finally, Facebook JavaScript[14] (FBJS) permits developers to use it in their applications. All the above tools give an open API to the developer for easy creation of Web applications that live inside Facebook and which are freely available to every Facebook user.

From Facebook to FaceBot. To exploit a social site, like Facebook, for launching DoS attacks, the adversary needs to create a malicious application, which embeds URIs to a victim Web server. These URIs must point to documents hosted by the victim, like images, text files, HTML pages, *etc.* When a user interacts with the application, the victim host will receive unsolicited requests. These requests are triggered through Facebook, since the application lives inside the social network, but they are actually generated by the Web browsers used by the users that access the malicious application. We define as *FaceBot* the collection of the users' Web browsers that are forced to generate requests upon viewing a malicious Facebook application. Schematically, a FaceBot is presented in Figure 9. The cloud groups a collection of Facebook users who browse a malicious application in Facebook. This causes a series of requests to be generated and directed towards the victim.

One crucial thing to note is that the application is hosted by the developer. That means that if an adversary wants to develop a malicious application, they must also host it. In other words, the adversary has to be able to cope with requests from users that are accessing the application. However, this can be overcomed using a free hosting service, specifically designed for Facebook applications.[15] But even if such a service were not available, the adversary has

[11] `http://www.facebook.com/developers/`
[12] `http://wiki.developers.facebook.com/index.php/FBML`
[13] `http://wiki.developers.facebook.com/index.php/FQL`
[14] `http://wiki.developers.facebook.com/index.php/FBJS`
[15] Joyent Free Accelerator:
`http://joyent.com/developers/facebook/`

to cope with much less traffic than the one that targets the victim. We further discuss this issue in Section 5.

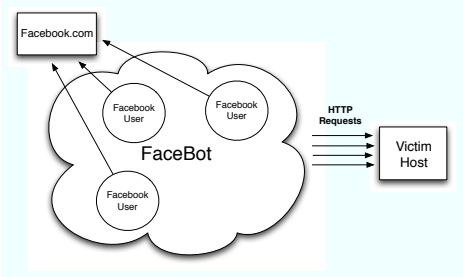

Fig. 9. The architecture of a FaceBot. Users access a malicious application in the social site (`facebook.com`) and subsequently a series of HTTP requests are created, which target the victim host.