DOI:10.1145/1498765.1498782

Web-based malware attacks are more insidious than ever. What can be done to stem the tide?

BY NIELS PROVOS, MOHEEB ABU RAJAB, AND PANAYIOTIS MAVROMMATIS

Cybercrime 2.0: When the Cloud Turns Dark

AS THE WEB has become vital for our day-to-day transactions, it has also become an attractive avenue for cyber crime. Financially motivated, the crime we see on the Web today is quite different from the more traditional network attacks. A few years ago adversaries heavily relied on remotely exploiting servers identified by scanning the Internet for vulnerable network services. Autonomously spreading computer worms such as Code Red and SQL Slammer were examples of such scanning attacks. Their huge scale put even the Internet at large at risk; for example, SQL Slammer generated traffic sufficient to melt down backbones. As a result, academia and industry alike developed effective ways to fortify the network perimeter against such attacks. Unfortunately, adversaries similarly changed tactics moving away from noisy scanning to more stealthy attacks.

Not only did they change their tactics, but also their motivation. Previously, large-scale events such as network worms were mostly exhibitions of technical superiority. Today, adversaries are primarily motivated by economic incentives to not only exploit and seize control of compromised systems for as long as possible but to turn their assets into revenue.

The Web offers adversaries a powerful infrastructure to compromise computer systems and monetize the resulting computing resources as well as any information that can be stolen from them. Adversaries achieve this by employing the Web to serve malicious Web content capable of compromising users' computers and running arbitrary code on them. This has largely been enabled due to the increased complexity of Web browsers and the resulting vulnerabilities that come with complex software. For example, a modern Web browser provides a powerful computing platform with access to different scripting languages, (for example, Javascript) as well as external plugins that may not follow the same security policies applied by the browser (for example, Flash, Java). While these capabilities enable sophisticated Web applications, they also allow adversaries to collect information about the target system and deliver exploits specifically tailored to a user's computer. Web attacks render perimeter defenses that disallow incoming connections useless against exploitation as adversaries use the browser to initiate out-bound connections to download attack payloads. This type of traffic looks almost identical to the users' normal browsing traffic and is not usually blocked by network firewalls.

To prevent Web-based malware from infecting users, Google has developed an infrastructure to identify malicious Web pages. The data resulting from this infrastructure is used to secure Web search results as well as protect browsers such as Firefox and Chrome. In this article, we discuss interesting Web attack trends as well as

some of the open challenges associated with this rising threat.

Web Attacks

As Web browsers have become more capable and the Web richer in features, it is difficult for the average user to understand what happens when visiting a Web page. In most applications visiting a Web page causes the browser to pull content from a number of different providers, for example, to show third-party ads, interactive maps, or display online videos. The shear number of possibilities to design Web pages and make them attractive to users is staggering. Overall, these features increase the complexity of the components that constitute a modern Web browser. Unfortunately, each browser component may introduce new vulnerabilities an adversary can leverage to gain control over a user's computer. Over the past few years we have seen an increasing number of browser vulnerabilities,[5,8] some of which have not had official fixes for weeks.

For an adversary to exploit a vulnerability, it requires the user visit a Web page that contains malicious content. One way to attract user traffic is to send spam email messages that advertise links to malicious Web pages. However, this delivery mechanism has some drawbacks. For the exploit to be delivered, the user must open the spam email and then click on the embedded link. The ubiquitous Web infrastructure provides a better solution to this bottleneck. While it is easy to exploit a Web browser, it is even easier to exploit Web servers. The relative simplicity of setting up and deploying Web servers has resulted in a large number of Web applications with remotely exploitable vulnerabilities. Unfortunately, these vulnerabilities are rarely patched, and therefore, remote exploitation of Web servers is increasing. To exploit users, adversaries just need to compromise a Web server and inject malicious content, for example, via an IFRAME pointing to an exploit server. Any visitor to such a compromised Web server becomes a target of exploitation. If the visitor's system is vulnerable, the exploit causes the browser to download and execute arbitrary payloads. We call this process "drive-by download." Depending on the popularity of the compromised Web site, an adversary may get access to a large user population. Last year, Web sites with millions of visitors were compromised that way.

Many drive-by downloads can be detected automatically via client honeypots. However, when adversaries use social engineering to trick the users into installing malicious software, automated detection is significantly complicated.

Taking Over Web Servers. Turning Web servers into infection vectors is unfortunately fairly straightforward. Over the last couple years, we have observed a number of different attacks against Web servers and Web applications, ranging from simple password guessing to more advanced attacks that can infect thousands of servers at once. In general, these attacks aim at altering Web site content to redirect visitors to servers controlled by the adversary. Here, we expand on some examples of recent dominant server attacks.

SQL Injection Attacks. SQL injection is an exploitation technique commonly used against Web servers that run vulnerable database applications. The vulnerability happens when user input is not properly sanitized (for example, by filtering escape characters and string literals) therefore causing well crafted user input to be interpreted as code and executed on the server. SQL injection has been commonly used to perpetrate unauthorized operations on a vulnerable database server such as harvesting users' information and manipulating the contents of the database. In Web applications running a SQL database to manage users' authentication, adversaries use SQL injection to bypass login and gain unauthorized access to user accounts or, even worse, to gain administrative access to the Web application. Other variants of these attacks allow the adversary to directly alter the contents of the server's database and inject the adversary's own content.

Last year, a major SQL injection attack was launched by the Asprox botnet.[15] In this attack several thousand bots were equipped with an SQL injection kit that starts by sending specially crafted queries to Google searching for servers that run ASP.net, and then launches SQL injection attacks against the Web sites returned from those queries. In these attacks the bot sends an encoded SQL query containing the exploit payload (similar to the format shown here) to the target Web server.

http://www.victim-site.com/asp_application.asp?arg=<encoded sql query>

The vulnerable server decodes and executes the query payload which, in the

Asprox case, yields SQL code similar to the snippet shown in Figure 1. The decoded payload searches the Web server folders for unicode and ASCII files and injects an IFRAME or a script tag in them. The injected content redirects the Web site users to Web servers controlled by the adversary and therefore subjects them to direct exploitation.

We monitored the Asprox botnet over the past eight months, and observed bots getting instructions to refresh their lists of the domains to inject. Overall, we have seen 340 different injected domains. Our analysis of the successful injections revealed that approximately six million URLs belonging to 153,000 different Web sites were victims of SQL injection attacks by the Asprox botnet. While the Asprox botnet is no longer active, several victim sites are still redirecting users to the malicious domains. Because bots inject code in a non-coordinated manner, many Web sites end up getting multiple injections of malicious scripts over time.

Redirections via `.htaccess`. Even when the Web pages on a server are harmless and unmodified, a Web server may still direct users to malicious content. Recently, adversaries compromised Apache-based Web servers and altered the configuration rules in the `.htaccess` file. This configuration file can be used for access control, but also allows for selective redirection of URLs to other destinations. In our analysis of Web servers, we have found several incidents where adversaries installed `.htaccess` configuration files to redirect visitors to malware distribution sites, for example, to fake anti-virus sites as we discuss later.

One interesting aspect of `.htaccess` redirections is the attempt to hide the compromise from the site owner. For example, redirection can be conditional based on how a visitor reached the compromised Web server as determined by the HTTP Referer header of the incoming request. In the incidents we observed, the `.htaccess` rules were configured so that visitors arriving via search engines were redirected to a malware site. However, when the site owner typed the URL directly into the browser's location bar, the site would load normally as the Referer header was not set.

Figure 2 shows an example of a compromised `.htaccess` file. In this example, users visiting the compromised site via any of the listed search engines will be redirected to http://89.28.13.204/in.html?s=xx. Notice that the initial redirect is usually to an IP address that acts as a staging server and redirects users to a continuously changing set of domains. The staging server manages which users get redirected where. For example, the staging server may check whether the user has already visited the redirector and return an empty payload on any subsequent visit. We assume this is meant to make analysis and reproduction of the redirection chain more difficult. Adversaries also frequently rewrite the `.htaccess` file to point to different IP addresses. Removing the `.htaccess` without patching the original vulnerability or changing the server credentials will not solve the problem. Many Web masters attempted to delete the `.htaccess` and found a new one on their servers the next day.

Figure 1. A decoded snippet of the SQL injection query sent by Asprox bots.[13]

```
DECLARE @T VARCHAR(255),@C VARCHAR(255)
DECLARE Table _ Cursor CURSOR FOR  SELECT a.name,b.name
FROM sysobjects a,syscolumns b
WHERE a.id=b.id AND a.xtype='u'
AND (b.xtype=99 OR b.xtype=35
OR b.xtype=231 OR b.xtype=167)
OPEN Table _ Cursor FETCH NEXT FROM Table _ Cursor INTO @T,@C
WHILE(@@FETCH _ STATUS=0)
BEGIN EXEC('UPDATE ['+@T+']
SET ['+@C+']=RTRIM(CONVERT(VARCHAR(4000),['+@C+']))+'''''')
FETCH NEXT FROM Table _ Cursor INTO @T,@C
END CLOSE Table _ Cursor
DEALLOCATE Table _ Cursor
```

Taking Over Web Users. Once the adversaries have turned a Web server into an infection vector, visitors to that site are subjected to various exploitation attempts. In general, client exploits fall under two main categories: automated drive-by downloads and social engineering attacks.

Drive-by downloads. In this category, adversaries attempt to exploit flaws in either the browser, the operating system, or the browser's external plugins. A successful exploit causes malware to be delivered and executed on the user's machine without her knowledge or concent. For example, a popular exploit we encountered takes advantage of a vulnerability in Microsoft Data Access Components (MDACS) that allows arbitrary code execution on a user's computer.[7] A 20-line Javascript code snippet was enough to exploit this vulnerability and initiate a drive-by download.

Another popular exploit is due to a vulnerability in Microsoft Windows WebViewFolderIcon. The exploit Javascript uses a technique called heap spraying that creates a large number of Javascript string objects on the heap. Each Javascript string contains x86 machine code (shellcode) necessary to download and execute a binary on the exploited system. By spraying the heap, an adversary attempts to create a copy of the shellcode at a known location in memory and then redirects program execution to it.

Social engineering attacks. When drive-by downloads fail to compromise a user's machine, adversaries often employ social

Figure 2. A snippet from the .htacsess file of a compromised Apache server.[11]

```
RewriteEngine On
RewriteCond %{HTTP _ REFERER} .*google.*$ [NC,OR]
RewriteCond %{HTTP _ REFERER} .*aol.*$ [NC,OR]
RewriteCond %{HTTP _ REFERER} .*msn.*$ [NC,OR]
RewriteCond %{HTTP _ REFERER} .*altavista.*$ [NC,OR]
RewriteCond %{HTTP _ REFERER} .*ask.*$ [NC,OR]
RewriteCond %{HTTP _ REFERER} .*yahoo.*$ [NC]
RewriteRule .* http://89.28.13.204/in.html?s=xx [R,L]
```

engineering techniques to trick users into installing and running malware by themselves. Unfortunately, the Web is rich with deceptive content that lures users into downloading malware.

One common class of attacks includes images that resemble popular video players, along with a false warning that the computer is missing essential codecs for displaying the video, or that a newer version of the video player plugin is required to view it. Instead, the provided link is for downloading a trojan that, once installed, gives the adversary full control over the user's machine.

A more recent trick involves fake security scans. A specially crafted Web site displays fake virus scanning dialogs, along with animated progress bars and a list of infections presumably found on the computer. All the warnings are false and are meant to scare the user into believing their machine is infected. The Web site then offers a download as solution, which could be another trojan, or ask the user for a registration fee to perform an unnecessary clean-up of their machine.

We have observed a steady increase in fake anti-virus attacks: From July to October 2008, we measured an average of 60 different domains serving fake security products, infecting an average of 1,500 Web sites. In November and December 2008, the number of domains increased to 475, infecting over 85,000 URLs. At that time the Federal Trade Commission reported that more than one million consumers were tricked into buying these products, and a U.S. district court issued a halt and an asset freeze on some of the companies behind these fake products.[3] This does not appear to have been sufficient to stop the scheme. In January 2009, we observed over 450 different domains serving fake security products, and the number of infected URLs had increased to 148,000.

Malware activities on the user's machine. Whether a user was compromised by a social engineering attack or a successful exploit and drive-by download, once the adversaries have control over a user's machine, they usually attempt to turn their work into profit.

In prior work,[10] we analyzed the behavior of Web malware installed by drive-by downloads. In many cases, malware was equipped with key-loggers to spy on the user's activity. Often, a back door was installed, allowing the adversary to access the machine directly at a later point in time. More sophisticated malware turned the machine into a bot listening to remote commands and executing various tasks on demand. For example, common uses of botnets include sending spam email or harvesting passwords or credit cards. Botnets afford the adversary a degree of anonymity since spam email appears to be sent from a set of continuously changing IP addresses making it harder to blacklist them.

To help improve the safety of the Internet, we have developed an extensive infrastructure for identifying URLs that trigger drive-by downloads. Our analysis starts by inspecting pages in Google's large Web repository. While exhaustive inspection of each page is prohibitively expensive as the repository contains billions of pages, we have developed a lightweight system to identify candidate pages more likely to be malicious. The candidate pages are then subjected to more detailed analysis in a virtual machine allowing us to determine if visiting a page results in malicious changes to the machine itself. The lightweight analysis uses a machine-learning framework that can detect 90% of all malicious pages with a false positive rate of only 10^{-3}. At this false positive rate, the filter reduces the workload of the virtual machines from billions of pages to only millions. The URLs that are determined to be malicious are further processed into host-suffix path-prefix patterns. Since 2006, our system has been used to protect Google's search. Our data is also published via Google's Safe Browsing API to browsers such a Firefox, Chrome, and Safari. These browsers employ our data to prevent users from visiting harmful pages.

Challenges

Despite our efforts to make the Web safer for users, there are still a number of fundamental challenges requiring future work, including:

Securing Web Services. Establishing a presence on the Web, ranging from simple HTML pages to advanced Web applications, has become an easy process that enables even people with little technical knowledge to set up a Web service. However, maintaining such a service and keeping it secure is still difficult. Many Web application frameworks require programmers to follow strict security practices, such as sanitizing and escaping user input. Unfortunately, as this burden is put onto the programmer, many Web applications suffer from vulnerabilities that can be remotely exploited.[12, 14] For example, SQL injection attacks are enabled by a programmer neglecting to escape external input.

Popular Web applications such as bulletin boards or blogs release security updates frequently, but many administrators neglect to update their installations. Even the Web server software itself, such as Apache or IIS, is often out-of-date. In previous work,[10] we found over 38% of Apache installations and 40% of PHP installations in compromised sites to be insecure and out-of-date.

To avoid the compromising of Web applications, it is important to develop mechanisms to keep Web servers and Web applications automatically patched. Some Web applications already notify Web masters about security updates, but the process of actually installing security patches is often still manual and complicated.

It is difficult to be completely safe against drive-by downloads. All that is required for an adversary to gain control over your system is a single vulnerability. Any piece of software that is exposed to Web content and not up-to-date can become the weakest link.

Many browser plugins and add-ons, such as toolbars, do not provide automatic updates. Furthermore, system updates often require a restart after installation discouraging users from applying the security patches on time.

Even if a system was fully patched, the window of vulnerability for some software is often very large. According to Krebs, major browsers were unsafe for as long as 284 days in 2006, and for at least 98 days criminals actively used vulnerabilities for which no patches were available to steal personal and financial data from users.[5, 6] Although progress on providing fault isolation in browsers that may prevent vulnerabilities from being exploited has been made,[1, 4] a completely secure browser still needs to be developed.

Detecting Social Engineering At-

tacks. Many drive-by downloads can be detected automatically via client honeypots. However, when adversaries use social engineering to trick the users into installing malicious software, automated detection is significantly complicated. Although, user interactions can be simulated by the client honeypot, a fundamental problem is the user's expectation about the functionality of a downloaded application compared to what it actually does. In the video case described earlier, the user expected to watch a video. After downloading and installing such a trojan, nothing usually happens. This could warn the user that something is amiss and might result in the user trying to fix their system. However, there is no reason why the installed software could not also play a video leaving the user with no reasons to suspect that she was infected.

Similarly, in addition to extorting the user for money, some of the fake anti-virus software does actually have some detection capability for old malware. The question then is how to determine if a piece of software functions as advertised. In general, this problem is undecidable. For example, the popular Google toolbar allows a user to opt into receiving the pagerank of a visited page. This works by sending the current URL to Google and then returning the associated pagerank and displaying it in the browser. This functionality was desired by the user and a legitimate feature. However, a similar piece of software might not disclose its functionality and send all visited URLs to some ominous third party. In that case, we would label the software spyware.

Automated analysis[2,9] is more difficult when malicious activity is triggered only under certain conditions. For example, some banking trojans watch the URL in the browser window and overlay a fake input field only for specific banking Web sites. Automated tools may discover the overlay functionality, but if the trojan was to compare against one-way hashes of URLs determining which banks were targeted could be rather difficult.

Conclusion

Without doubt, Web-based malware is a security concern for many users. Unfortunately, the root cause that allows the Web to be leveraged for malware delivery is an inherent lack of security in its design—neither Web applications nor the Internet infrastructure supporting these applications were designed with a well-thought-out security model. Browsers evolved in complexity to support a wide range of applications and inherited some of these weaknesses and added more of their own. While some of the solutions in this space are promising and may help reduce the magnitude of the problem, safe browsing will continue to be a far sought-after goal that deserves serious attention from academia and industry alike. C

Whether a user was compromised by a social engineering attack, or a successful exploit and drive-by download, once the adversaries have control over a user's machine, they usually attempt to turn their work into profit.

References

1. Barth, A., Jackson, C., and Reis, C. *The Security Architecture of the Chromium Browser;* http://crypto.stanford.edu/websec/chromium/chromium-security-architecture.odf.
2. Brumley, D., Hartwig, C., Kang, M., Liang, Z., Newsome, J., Song, D., and Yin, H. BitScope: Automatically dissecting malicious binaries. Technical Report, Technical Report CMU-CS-07-133, School of Computer Science, Carnegie Mellon University, March 2007.
3. Court halts bogus computer scans (Dec. 2008); www.ftc.gov/opa/2008/12/winsoftware.shtm
4. Grier, C., Tang, S., and King, S. Secure Web browsing with the OP Web browser. *Security and Privacy*, 2008. IEEE Symposium, 2008, 402–416.
5. Krebs, B. Internet Explorer unsafe for 284 days in 2006. *Washington Post Online* Blog, Jan. 2007.
6. Krebs, B. Blogfight: IE vs. Firefox security. *Washington Post Online* Blog, Jan. 2009.
7. Microsoft. Microsoft Security Bulletin MS06-014: Vulnerability in the Microsoft Data Access Components (MDACS) Function Could Allow Code Execution. May, 2006.
8. Microsoft. Microsoft Security Advisory (935423): Vulnerability in Windows Animated Cursor Handling, Mar. 2007.
9. Moser, A., Kruegel, C., and Kirda, E. Exploring multiple execution paths for malware analysis. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2007, 231–245.
10. Polychronakis, M., Mavrommatis, P., and Provos, N. Ghost Turns Zombie: Exploring the Life Cycle of Web-based Malware. In *Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats* (Apr. 2008).
11. Provos, N. Using htaccess To Distribute Malware. Dec. 2008; www.provos.org/index.php?/archives/55-Using-htaccess-To-Distribute-Malware.html.
12. Provos, N., Mavrommatis, P., Rajab, M.A., and Monrose, F. All your IFRAMEs point to us. *USENIX Security Symposium*, 2008, 1–16.
13. Raz, R. Asprox silent defacement. *Chapters in Web Security*, Dec. 2008; http://chaptersinWebsecurity.blogspot.com/2008/07/asprox-silent-defacement.html.
14. Small, S., Mason, J., Monrose, F., Provos, N., and Stubblefield, A. To catch a predator: A natural language approach for eliciting malicious payloads. *USENIX Security Symposium*, 2008, 171–184.
15. Stewart, J. Danmec/Asprox SQL injection attack tool analysis. *Secure Works Online*, May 2008; www.secureworks.com/research/threats/danmecasprox.

Niels Provos (niels@google.com) joined Google in 2003 and is currently a principle software engineer in the Infrastructure Security Group. His areas of interest include computer and network security as well as large-scale distributed systems. He is serving on the USENIX Board of Directors.

Moheeb Abu Rajab (moheeb@google.com) joined Google in 2008 and is currently a software engineer in the Infrastructure Security Group. His areas of interest include computer and network security.

Panayiotis Mavrommatis (Panayiotis@google.com) joined Google in 2006 and is currently working as a senior software engineer in the Security Group.