Automated Web Patrol with Strider HoneyMonkeys: *Finding Web Sites That Exploit Browser Vulnerabilities*

Yi-Min Wang, Doug Beck, Xuxian Jiang, Roussi Roussev, Chad Verbowski, Shuo Chen, and Sam King
Microsoft Research, Redmond

Abstract

Internet attacks that use malicious web sites to install malware programs by exploiting browser vulnerabilities are a serious emerging threat. In response, we have developed an automated web patrol system to automatically identify and monitor these malicious sites. We describe the design and implementation of the Strider HoneyMonkey Exploit Detection System, which consists of a pipeline of "monkey programs" running possibly vulnerable browsers on virtual machines with different patch levels and patrolling the Web to seek out and classify web sites that exploit browser vulnerabilities.

Within the first month of utilizing this system, we identified 752 unique URLs hosted on 288 web sites that could successfully exploit unpatched Windows XP machines. The system automatically constructed topology graphs based on traffic redirection to capture the relationship between the exploit sites. This allowed us to identify several major players who are responsible for a large number of exploit pages. By monitoring these 752 exploit-URLs on a daily basis, we discovered a malicious web site that was performing zero-day exploits of the unpatched javaprxy.dll vulnerability and was operating behind 25 exploit-URLs. It was confirmed as the first "in-the-wild", zero-day exploit of this vulnerability that was reported to the Microsoft Security Response Center. Additionally, by scanning the most popular one million URLs as classified by a search engine, we found over seven hundred exploit-URLs, many of which serve popular content related to celebrities, song lyrics, wallpapers, video game cheats, and wrestling.

1. Introduction

Internet attacks that use a malicious or hacked web site to exploit unpatched client-side vulnerabilities of visiting browsers are on the rise. Malcode distributed by this method in the past 12 months includes the Download.Ject [D04], Bofra [R04], and Xpire.info [B04] programs. These attacks allow web servers that host compromised URLs to install malcode on visiting client machines without requiring any user interaction beyond visitation. There have been several manual analyses of these events [E04,F04,G05,IF05,R05,S05,T05]. Although these analyses provide very useful and detailed information about which vulnerabilities are exploited and which malware programs are installed, such efforts are not scalable, do not provide a comprehensive picture of the problem, and are generally ineffective at efficiently finding new malicious sites.

To address these issues, we developed a system that uses *a pipeline of active, client-side, Virtual Machine (VM)-based honeypots* [H,HC], called *Strider HoneyMonkeys*, to perform large-scale, systematic and automated web patrol. The HoneyMonkey system uses monkey programs[1] that run within virtual machines with OS's of various patch levels to drive web browsers in an attempt to mimic human web browsing. Our approach adopts a state-management methodology to cybersecurity: instead of directly detecting the acts of vulnerability exploits, the system uses the Strider Tracer [W03] to catch unauthorized file creations and configuration changes that are the result of a successful exploit.

We demonstrate the effectiveness of our method by discovering a large community of malicious web sites that host exploit pages and by deriving the redirection relationships among them. We describe a real-world experience with identifying a zero-day exploit[2] using this system. We show the existence of hundreds of malicious web pages amongst many popular web sites. Finally, we propose a comprehensive anti-exploit process based on this monitoring system in order to improve Internet safety.

This paper is organized as follows. Section 2 provides background information on the problem space by describing the techniques used in actual client-side exploits of popular web browsers. Section 3 gives an overview of the Strider HoneyMonkey Exploit Detection System and its surrounding Anti-Exploit Process. Section

[1] An automation-enabled program such as the Internet Explorer browser allows programmatic access to most of the operations that can be invoked by a user. A "monkey program" is a program that drives the browser in a way that mimics a human user's operation.

[2] In this paper, a zero-day exploit refers to a vulnerability exploit that exists before the patch for the vulnerability is released. The vulnerability can be known or unknown to the public at that time.

4 evaluates the effectiveness of HoneyMonkey in both known-vulnerability and zero-day exploit detection, and presents an analysis of the exploit data to help prioritize investigation tasks. Section 5 discusses the limitations of and possible attacks on the current HoneyMonkey system and describes several countermeasures including an enhancement based on a vulnerability-specific exploit detection mechanism. Section 6 surveys related work and Section 7 concludes the paper.

2. Browser-based Vulnerability Exploits

Malicious activities performed by actual web sites exploiting browser vulnerabilities can be divided into four steps: code obfuscation, URL redirection, vulnerability exploitation, and malware installation.

2.1. Code Obfuscation

To complicate investigation and to escape signature-based scanning by anti-virus/anti-spyware software, some web sites use a combination of the following code obfuscation techniques: (1) dynamic code injection using the `document.write()` function inside a script; (2) unreadable, long strings with encoded characters such as "%28", "&#104", etc. which are then decoded either by the `unescape()` function inside a script or by the browser; (3) custom decoding routine included in a script; and (4) sub-string replacement using the `replace()` function. Since code-obfuscation is a common technique, this limits the ability of attack-signature-based detectors to detect new attacks that leverage old exploit code.

2.2. URL Redirection

Most malicious web sites automatically redirect browser traffic to additional URLs. Specifically, when a browser visits a primary URL, the response from that URL instructs the browser to automatically visit one or more secondary URLs, which may or may not affect the content that is displayed to the user. Such redirections typically use one of the following mechanisms classified into three categories: (1) protocol redirection using HTTP 302 Temporary Redirect; (2) HTML tags including `<iframe>`, `<frame>` inside `<frameset>`, and `<META http-equiv=refresh>`; (3) script functions including `window.location.replace()`, `window.location.href()`, `window.open()`, `window.showModalDialog()`, and `<link_ID>.click()`, etc. Since redirection is commonly used by non-malicious sites to enrich content, simply eliminating redirection from a browser would present significant complications

2.3. Vulnerability Exploitation

It is not uncommon to see a malicious web page attempting to exploit multiple browser vulnerabilities in order to maximize the chance of a successful attack. Figure 1 shows an example HTML fragment that uses various primitives to load multiple files from different URLs on the same server to exploit three vulnerabilities fixed in Microsoft Security Bulletins MS05-002 [M52], MS03-011 [M311], and MS04-013 [M413]. If any of the exploits succeeds, a Trojan downloader named *win32.exe* is downloaded and executed. Note that although Internet Explorer is the common target due to its popularity, other browsers can also be attacked.

2.4. Malware Installation

The purpose of an exploit is almost always to introduce some piece of arbitrary code on the victim machine, as a way to achieve a larger attack goal. We have observed a plethora of malcode types installed through browser exploits, including *viruses* that infect files, *backdoors* that open entry points for future unauthorized access, *bot* programs that allow the attacker to control a whole network of compromised systems, *Trojan downloaders* that connect to the Internet and download other programs, *Trojan droppers* that drop files from themselves without accessing the Internet, and *Trojan proxies* that redirect network traffic. Some spyware

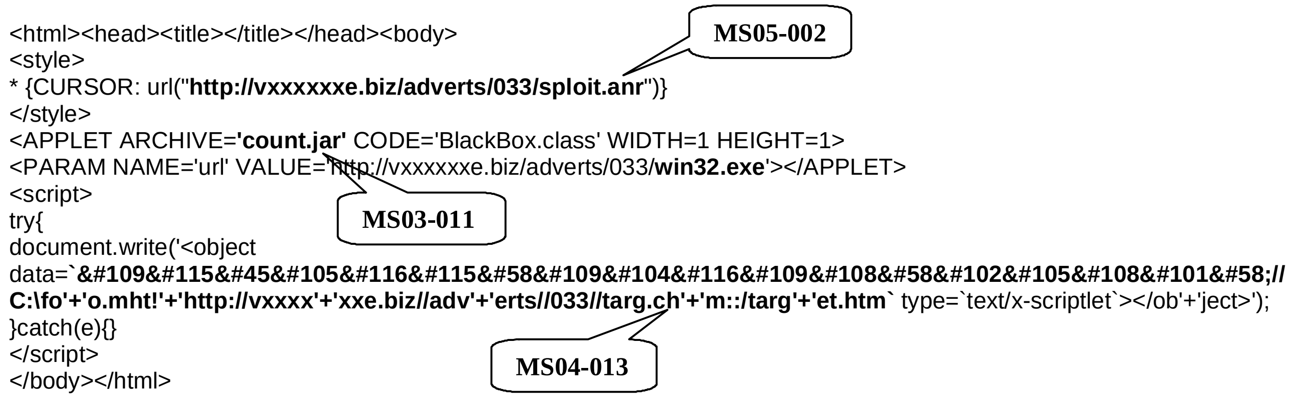
```
<html><head><title></title></head><body>
<style>
* {CURSOR: url("http://vxxxxxxe.biz/adverts/033/sploit.anr")}
</style>
<APPLET ARCHIVE='count.jar' CODE='BlackBox.class' WIDTH=1 HEIGHT=1>
<PARAM NAME='url' VALUE='http://vxxxxxxe.biz/adverts/033/win32.exe'></APPLET>
<script>
try{
document.write('<object
data=`&#109&#115&#45&#105&#116&#115&#58&#109&#104&#116&#109&#108&#58&#102&#105&#108&#101&#58;//
C:\fo'+'o.mht!'+'http://vxxxx'+'xxe.biz//adv'+'erts//033//targ.ch'+'m::/targ'+'et.htm` type=`text/x-scriptlet`></ob'+'ject>');
}catch(e){}
</script>
</body></html>
```
MS05-002
MS03-011
MS04-013

Figure 1. Actual sample Web page attempting to exploit multiple vulnerabilities

programs and even anti-spyware programs are also installed through exploits.

3. The HoneyMonkey System

The HoneyMonkey system attempts to automatically detect and analyze a network of web sites that exploit web browsers. Figure 2 illustrates the HoneyMonkey Exploit Detection System, shown inside the dotted square, and the surrounding Anti-Exploit Process which includes both automatic and manual components.

3.1. Exploit Detection System

The exploit detection system is the heart of the HoneyMonkeys design. This system consists of a 3-stage pipeline of virtual machines. Given a large list of input URLs with a potentially low exploit-URL density, each HoneyMonkey in Stage 1 starts with a *scalable mode* by visiting *N* URLs simultaneously inside one unpatched VM. When the HoneyMonkey detects an exploit, it switches to the *basic*, *one-URL-per-VM mode* to re-test each of the *N* suspects in order to determine which ones are exploit URLs.

Stage-2 HoneyMonkeys scan Stage 1 detected exploit-URLs and perform recursive redirection analysis to identify all web pages involved in exploit activities and to determine their relationships. Stage-3 HoneyMonkeys continuously scan Stage-2 detected exploit-URLs using (nearly) fully patched VMs in order to detect attacks exploiting the latest vulnerabilities.

We used a network of 20 machines to produce the results reported in this paper. Each machine had a CPU speed between 1.7 and 3.2 GHz, a memory size between 512 MB and 2GB, and was responsible for running one VM configured with 256 MB to 512MB of RAM. Each VM supported up to 10 simultaneous browser processes in the scalable mode, with each process visiting a different URL. Due to the way HoneyMonkeys detect exploits (discussed later), there is a trade-off between the scan rate and the robustness of exploit detection: if the HoneyMonkey does not wait long enough or if too many simultaneous browser processes cause excessive slowdown, some exploit pages may not be able to perform a detectable attack (e.g., beginning a software installation).

Through extensive experiments, we determined that a wait time of two minutes was a good trade-off. Taking into account the overhead of restarting VMs in a clean state, each machine was able to scan and analyze between 3,000 to 4,000 URLs per day. We have since improved the scalability of the system to a scan rate of 8,000 URLs per day per machine in the scalable mode. (In contrast, the basic mode scans between 500 and 700 URLs per day per machine.) We expect that using a more sophisticated VM platform that enables significantly more VMs per host machine and faster rollback [VMC+05] would significantly increase our scalability.

3.1.1. Exploit Detection

Although it is possible to detect browser exploits by building signature-based detection code for each known vulnerability or exploit, this approach is manually intensive. To lower this cost, we take the following *black-*

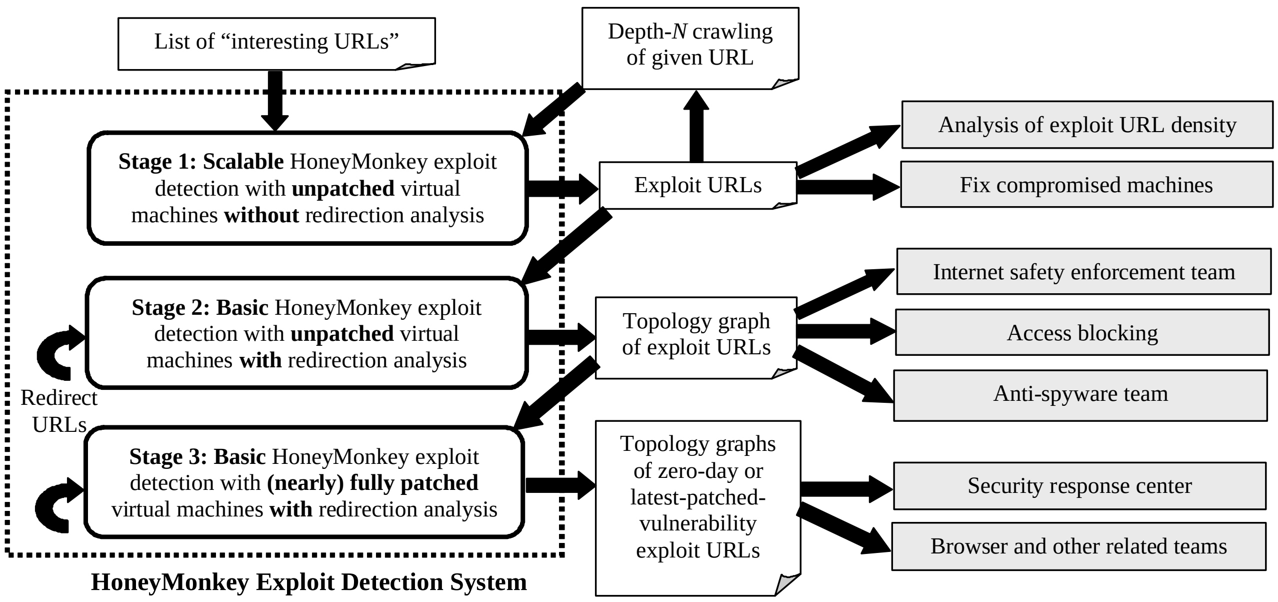
List of "interesting URLs"
Depth-N crawling of given URL
Stage 1: Scalable HoneyMonkey exploit detection with unpatched virtual machines without redirection analysis
Exploit URLs
Analysis of exploit URL density
Fix compromised machines
Stage 2: Basic HoneyMonkey exploit detection with unpatched virtual machines with redirection analysis
Topology graph of exploit URLs
Internet safety enforcement team
Access blocking
Anti-spyware team
Redirect URLs
Stage 3: Basic HoneyMonkey exploit detection with (nearly) fully patched virtual machines with redirection analysis
Topology graphs of zero-day or latest-patched-vulnerability exploit URLs
Security response center
Browser and other related teams
HoneyMonkey Exploit Detection System

Figure 2. HoneyMonkey Exploit Detection System and Anti-Exploit Process

box, non-signature-based approach: we run a monkey program that launches a browser instance to visit each input URL and then waits for a few minutes to allow downloading of any code which may have a short time delay. We then detect a group of persistent-state changes to signal exploitation. Since the monkey is not instructed to click on any dialog box to permit software installation, any executable files or registry entries created outside the browser sandbox indicate an exploit. This approach has the additional important advantage of *allowing the detection of known-vulnerability exploits and zero-day exploits in a uniform way*. Specifically, the same monkey program running on unpatched machines to detect a broad range of browser-based vulnerability exploits (as shown in Stages 1 and 2) can run on fully patched machines to detect zero-day exploits, as shown in Stage 3.

At the end of each visit, the HoneyMonkey generates an XML report containing the following five pieces of information:

(1) Executable files created or modified outside the browser sandbox folders: this is the primary mechanism for exploit detection. It is implemented on top of the Strider Tracer [W03], which uses a file-tracing driver to efficiently record every single file read/write operation.

(2) Processes created: Strider Tracer also tracks all child processes created by the browser process.

(3) Windows registry entries created or modified: Strider Tracer additionally includes a driver that efficiently records every single registry [G04] read/write. To highlight the most critical entries, we use the Strider Gatekeeper and GhostBuster filters [W04,W05], which target registry entries most frequently attacked by spyware, Trojans, and rootkits based on an extensive study. This allows HoneyMonkey to detect exploits that modify critical configuration settings (such as the browser home page and the wallpaper) without creating executable files.

(4) Vulnerability exploited: to provide additional information and to address limitations of the black-box approach, we have developed and incorporated a vulnerability-specific detector, to be discussed in Section 5. This is based on the *vulnerability* signature of the exploit, rather than on any particular piece of malcode.

(5) Redirect-URLs visited: Since malcode is often laundered through other sites, this module allows us to track redirections to determine both the real source of the malcode and those involved in the distribution chain.

To ease cleanup of infected state, we run HoneyMonkeys inside a VM. (Our current implementation uses Microsoft Virtual PC and Virtual Server.) Upon detecting an exploit, the monkey saves its logs and notifies the ***Monkey Controller*** on the host machine to destroy the infected VM and re-spawn a clean HoneyMonkey, which then continues to visit the remaining URL list. The Monkey Controller then passes the detected exploit-URL to the next monkey in the pipeline to further investigate the strength of the exploit.

3.1.2. Redirection Analysis

Many exploit-URLs identified in Stage 1 do not perform the actual exploits but instead act as front-end ***content providers*** that serve "interesting" content such as pornography in order to attract browser traffic. This traffic is then sold and redirected to back-end ***exploit providers***, which specialize in exploiting clients and installing malware.

URLs visited through traffic redirection can be tracked with a Browser Helper Object (BHO) running within each browser process or by intercepting and analyzing network packets. When the HoneyMonkey runs in its "redirection analysis" mode, any automatically visited URLs are fed back to the system for further checking. This recursive scanning allows the construction of ***topology graphs based on traffic redirection***. In Section 4, we present our analysis of topology graphs to demonstrate how they enable the identification of major exploit providers that receive traffic from a large number of content providers; they also show how exploit providers organize their web pages in a way that facilitates customized malware installations for each of their affiliates. Finally, we are able to positively identify the web pages that actually perform the exploits by implementing an option in our redirection tracker to block all redirection traffic.

3.2. Anti-Exploit Process

The Anti-Exploit Process involves generating the input URL lists for HoneyMonkeys to scan, and taking various actions based on analyses of the output exploit-URL data.

3.2.1. Generating Input URL Lists

We use three sources for generating "interesting" URLs for analysis. The first category consists of suspicious URLs including web sites that are known to host spyware [CWS05] or malware, links appearing in phishing or spam emails [S05] or instant messages, web pages serving questionable content such as pornography, URL names that are typos of popular sites [G05], web sites involved in DNS cache poisoning [HD05,IW05,S04], and similar common sources of malicious web content.

The second category consists of the most popular web pages, which, if compromised, can potentially infect a large population. Examples include the top 100,000 web sites based on browser traffic ranking [AL] or the top *N* million web sites based on click-through counts as measured by search engines.

The third category encompasses URL lists of a more localized scope. For example, an organization may want to regularly verify that its web pages have not been compromised to exploit visitors; a user may want to investigate whether any recently visited URL was responsible for causing a spyware infection.

3.2.2. Acting on Output Exploit-URL Data

Stage 1 Output – Exploit-URLs

The percentage of exploit-URLs in a given list can be used to measure the risk of web surfing. For example, by comparing the percentage numbers from two URL lists corresponding to two different search categories (e.g., gambling versus shopping), we can assess the relative risk of malware infection for people with different browsing habits. Also, we have observed that depth-*N* crawling of exploit pages containing a large number of links, as illustrated at the top of Figure 2, often leads to the discovery of more exploit pages.

Stage 2 Output – Traffic-Redirection Topology Graphs

The HoneyMonkey system currently serves as a lead-generation tool for the Internet safety enforcement team in the Microsoft legal department. The topology graphs and subsequent investigations of the malicious behavior of the installed malware programs provide a prioritized list for potential enforcement actions that include sending site-takedown notices, notifying law enforcement agencies, and filing civil suits against the individuals responsible for distributing the malware programs. We have successfully shut down several malicious URLs discovered by the HoneyMonkey.

Due to the international nature of the exploit community, access blocking may be more appropriate and effective than legal actions in many cases. Blocking can be implemented at different levels: search engines can remove exploit-URLs from their database; Internet Service Providers (ISPs) can black-list exploit-URLs to protect their entire customer base; corporate proxy servers can prevent employees from accessing any of the exploit-URLs; and individual users can block their machines from communicating with any exploit sites by editing their local "hosts" files to map those server hostnames to a local loopback IP address.

Exploit-URLs also provide valuable leads to our anti-spyware product team. Each installed program is tagged with an "exploit-based installation without user permission" attribute. This clearly distinguishes the program from other more benign spyware programs that are always installed after a user accepts the licensing agreement.

Stage 3 Output – Zero-Day Exploit-URLs and Topology Graphs

By constantly monitoring all known exploit-URLs using HoneyMonkeys running on fully patched machines, we can detect zero-day exploits either when one of the monitored URLs "upgrade" its own exploit code or when a new URL that hosts zero-day exploit code starts receiving redirection traffic from any of the monitored URLs. Zero-day exploit monitoring is perhaps the most valuable contribution of the HoneyMonkey because zero-day exploits can be extremely damaging and whether they are actually being used in the wild is the most critical piece of information in the decision process for security guidance, patch development, and patch release. When a HoneyMonkey detects a zero-day exploit, it reports the URL to the Microsoft Security Response Center, which works closely with the enforcement team and the groups owning the software with the vulnerability to thoroughly investigate the case and determine the most appropriate course of action. We will discuss an actual case in Section 4.2.

Due to the unavoidable delay between patch release and patch deployment, it is important to know whether the vulnerabilities fixed in the newly released patch are being actively exploited in the wild. Such latest-patched-vulnerability exploit monitoring can be achieved by running HoneyMonkeys on nearly fully patched machines, which are missing only the latest patch. This provides visibility into the prevalence of such exploits to help provide guidance on the urgency of patch deployment.

4. Experimental Evaluation

We present experimental results in three sections: scanning suspicious URLs, zero-day exploit detection, and scanning popular URLs. We refer to the first and the third sets of data as *"suspicious-list data"* and *"popular-list data"*, respectively. All experiments were performed with Internet Explorer browser version 6.0.

We note that the statistics reported in this paper do not allow us to calculate the total number of end-hosts exploited by the malicious web sites we have found. Such calculations would require knowing precisely the number of machines that have visited each exploit page and whether each machine has patched the specific vulnerabilities targeted by each visited exploit page.

4.1. Scanning Suspicious URLs

4.1.1. Summary Statistics

Our first experiment aimed at gathering a list of most likely candidates for exploit-URLs in order to get the highest hit rate possible. We collected 16,190 potentially malicious URLs from three sources: (1) a web search of "known-bad" web sites involved in the installations of malicious spyware programs [CWS05]; (2) a web search for Windows "hosts" files [HF] that are used to block advertisements and bad sites by controlling the domain name-to-IP address mapping; (3) depth-2 crawling of some of the discovered exploit-URLs.

We used the Stage-1 HoneyMonkeys running on unpatched WinXP SP1 and SP2 VMs to scan the 16,190 URLs and identified 207 as exploit-URLs; this translates into a density of ***1.28%***. This serves as an upper bound on the infection rate: if a user does not patch his machine at all and he exclusively visits risky web sites with questionable content, his machine will get exploited by approximately one out of every 100 URLs he visits. We will discuss the exploit-URL density for normal browsing behavior in Section 4.3.

After recursive redirection analysis by Stage-2 HoneyMonkeys, the list expanded from 207 URLs to 752 URLs – a ***263%*** expansion. This reveals that there is a sophisticated network of exploit providers hiding behind URL redirection to perform malicious activities.

Figure 3 shows the breakdown of the 752 exploit-URLs among different service-pack (SP1 or SP2) and patch levels, where "***UP***" stands for "UnPatched", "***PP***" stands for "Partially Patched", and "***FP***" stands for "Fully Patched". As expected, the SP1-UP number is much higher than the SP2-UP number because the former has more known vulnerabilities that have existed for a longer time.

	Number of Exploit-URLs	**Number of Exploit Sites**
Total	752	288
SP1 Unpatched (SP1-UP)	688	268
SP2 Unpatched (SP2-UP)	204	115
SP2 Partially Patched (SP2-PP)	17	10
SP2 Fully Patched (SP2-FP)	0	0

Figure 3. Exploit statistics for Windows XP as a function of patch levels (May/June 2005 data)

The SP2-PP numbers are the numbers of exploit pages and sites that successfully exploited a WinXP SP2 machine partially patched up to early 2005. The fact that the numbers are one order of magnitude lower than their SP2-UP counterparts demonstrates the importance of patching. An important observation is that only a small percentage of exploit sites are updating their exploit capabilities to keep up with the latest vulnerabilities, even though proof-of-concept exploit code for most of the vulnerabilities are publicly posted. We believe this is due to three factors: (1) Upgrading and testing new exploit code incurs some cost which needs to be traded off against the increase in the number of victim machines; (2) Some vulnerabilities are more difficult to exploit than others; for example, some of the attacks are nondeterministic or take longer. Most exploiters tend to stay with existing, reliable exploits, and only upgrade when they find the next easy target. (3) Most security-conscious web users diligently apply patches. Exploit sites with "advanced" capabilities are likely to draw attention from knowledgeable users and become targets for investigation.

The SP2-FP numbers again demonstrate the importance of software patching: none of the 752 exploit-URLs was able to exploit a fully updated WinXP SP2 machine according to our May/June 2005 data. As we describe in Section 4.2, there was a period of time in early July when this was no longer true. We were able to quickly identify and report the few exploit providers capable of infecting fully patched machines, which led to actions to shut them down.

4.1.2. Topology graphs and node ranking

Figure 4 shows the topology graph of the 17 exploit-URLs for SP2-PP. These are among the most powerful exploit pages in terms of the number of machines they are capable of infecting and should be considered high priorities for investigation. Rectangular nodes represent individual exploit-URLs. Solid arrows between rectangles represent automatic traffic redirection. Circles represent *site nodes* that act as an aggregation point for all exploit pages hosted on that site, with the site node having a thin edge connecting each of its child-page rectangles. Nodes that do not receive redirected traffic are most likely content providers. Nodes that receive traffic from multiple exploit sites (for example, the large rectangle **R** at the bottom) are most likely exploit providers.

The size of a node is proportional to the number of cross-site arrows directly connected to it, both incoming and outgoing. Such numbers provide a good indication of the relative popularity of exploit-URLs and sites and are referred to as ***connection counts***. It is clear from the picture that the large rectangle **R** and its associated circle **C** have the highest connection counts. Therefore, blocking access to this site would be the most effective starting point since it would disrupt nearly half of this exploit network.

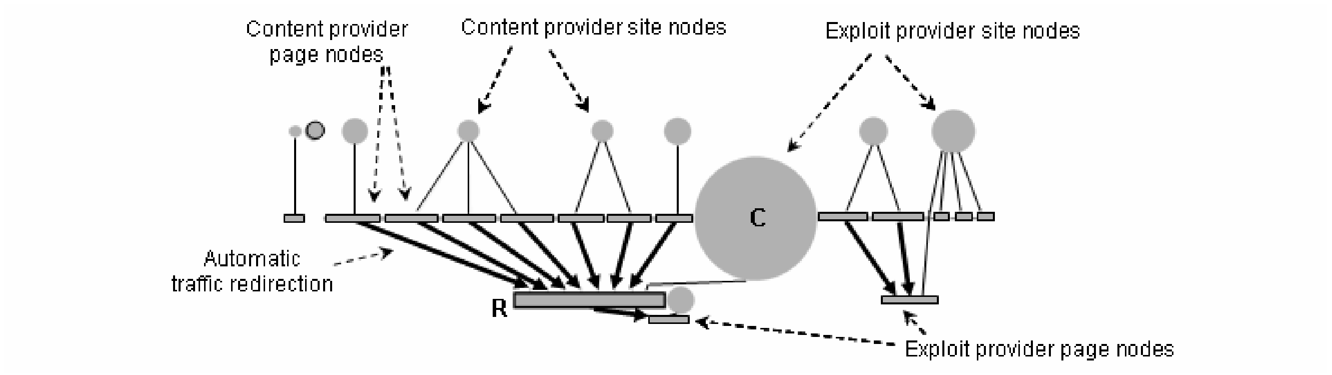
Content provider page nodes
Content provider site nodes
Exploit provider site nodes
C
Automatic traffic redirection
R
Exploit provider page nodes

Figure 4. SP2-PP topology graph (17 URLs, 10 sites)

The topology graph for the 688 SP1-UP exploit-URLs is much larger and more complex. It is only useful when viewed from a graph manipulation tool and is therefore omitted here. Most of the URLs appear to be pornography pages and the aggressive traffic redirection among them leads to the complexity of the bulk of the graph. In the isolated corners, we found a shopping site redirecting traffic to five advertising companies that serve exploiting advertisements, a screensaver freeware site, and over 20 exploit search sites. Next, we describe two ranking algorithms that help prioritize the investigations of these hundreds of URLs and sites.

<u>Site ranking based on connection counts</u>

Figure 5 illustrates the top 15 exploit sites for SP1-UP according to their connection counts. The bar height indicates how many other sites a given site has direct traffic-redirection relationship with and likely reflects how entrenched a site owner is with the exploit community. The bar for each site is composed of three segments of different colors: a black segment represents the number of sites that redirect traffic here; a white segment represents the number of sites to which traffic is redirected; a gray segment indicates the number of sites that have two-way traffic redirection relationship with the given site.

For example, site #15 corresponds to a content provider who is selling traffic to multiple exploit providers and sharing traffic with a few other content providers. Site #7 corresponds to an exploit provider that is receiving traffic from multiple web sites. Sites #4, #5, and #9 correspond to pornography sites that play a complicated role: they redirect traffic to many exploit providers and receive traffic from many content providers. Their heavy involvement in exploit activities and the fact that they are registered to the same owner suggest that they may be set up primarily for exploit purposes.

Site ranking, categorization, and grouping play a key role in the anti-exploit process because it serves as the basis for deciding the most effective resource allocation for monitoring, investigation, blocking, and legal actions. For example, high-ranked exploit sites in Figure 5 should be heavily monitored because a zero-day exploit page connected to any of them would likely affect a large number of web sites. Legal investigations should focus on top exploit providers, rather than content providers that are mere traffic redirectors and do not perform exploits themselves.

<u>Site ranking based on number of hosted exploit-URLs</u>

Figure 6 illustrates the top 129 sites, each hosting more than one exploit URL. This ranking helps highlight those web sites whose internal page hierarchy provides important insights. First, some web sites host a large number of exploit pages with a well-organized hierarchical structure. For example, the #1 site hosts 24 exploit pages that are organized by what look likes account names for affiliates; many others organize their exploit pages by affiliate IDs or referring site names; some even organize their pages by the names of the vulnerabilities they exploit and a few of them have the word “exploit” as part of the URL names.

The second observation is that some sophisticated web sites use transient URLs that contain random strings. This is designed to make investigations more difficult. Site ranking based on the number of hosted exploit-URLs helps highlight such sites so that they are prioritized higher for investigation. The zero-day exploits discussed in the next sub-section provide a good example of this.

4.2. Zero-Day Exploit Detection

In early July 2005, a Stage-3 HoneyMonkey discovered our first zero-day exploit. The javaprxy.dll vulnerability was known at that time without an available patch [J105,J205], and whether it was actually being exploited was a critical piece of information that was previously not known. The HoneyMonkey system detected the first exploit page within 2.5 hours of scanning and it was confirmed to be the first in-the-wild exploit-URL of the vulnerability reported to the Microsoft Security Response Center. A second exploit-URL was

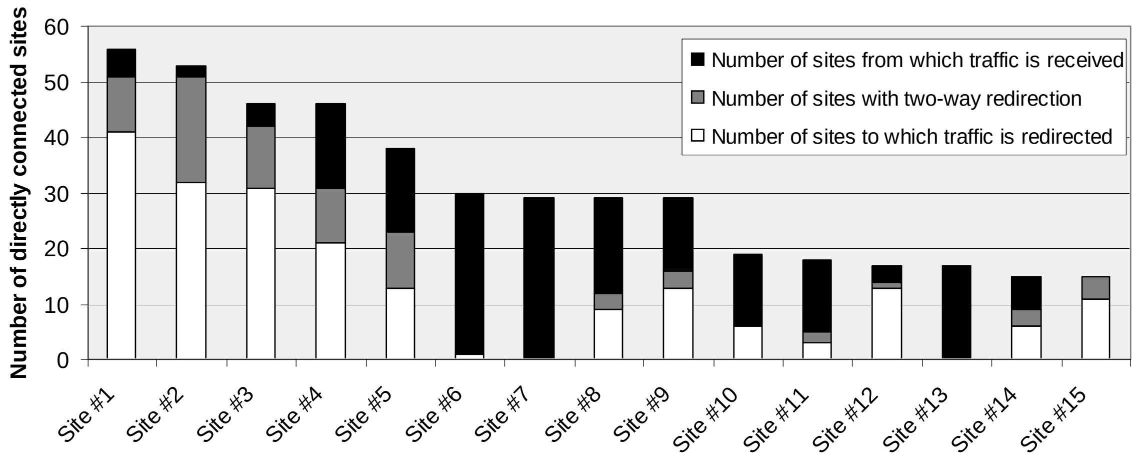

Figure 5. Top 15 exploit sites ranked by connection counts, among the 268 SP1-UP exploit sites from the suspicious list

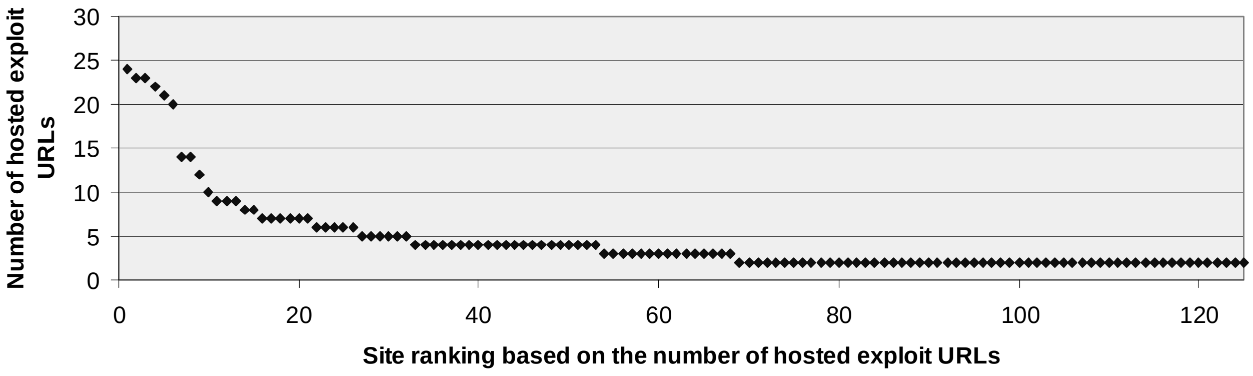

Figure 6. Top 129 SP1-UP exploit sites ranked by the number of exploit-URLs hosted

detected in the next hour. These two occupy positions #132 and #179, respectively, in our list of 752 monitored URLs. This information enabled the response center to provide customers with a security advisory and a follow-up security bulletin [SH, J205].

During the subsequent five days, HoneyMonkey detected that 26 of the 752 exploit-URLs upgraded to the zero-day exploit. Redirection analysis further revealed that 25 of them were redirecting traffic to a previously unknown exploit provider site that was hosting exploit-URLs with names in the following form:

http://[IP address]/[8 chars]/test2/iejp.htm

where [8 chars] consists of 8 random characters that appeared to change gradually over time. Takedown notices were sent after further investigation of the installed malware programs, and most of the 25 web pages stopped exploiting the javaprxy.dll vulnerability shortly after that.

Latest-Patched-Vulnerability Exploit Monitoring

One day after the patch release, HoneyMonkey detected another jump in the number of exploit-URLs for the vulnerability: 53 URLs from 12 sites were upgraded in the subsequent six days. Redirection analysis revealed that all of them were redirecting traffic to a previously known exploit provider (ranked #1 in Figure 6) who decided to add a new exploit page for javaprxy.dll to increase its infection base. A takedown notice was sent after malware investigation and all 53 URLs stopped exploiting within a couple of days.

Important Observations

This experience provides concrete evidence that the HoneyMonkey system can potentially evolve into a full-fledged, systematic and automatic zero-day exploit monitoring system for browser-based attacks. We make the following observations from the initial success:

(1) Monitoring easy-to-find exploit-URLs is effective: we predicted that monitoring the 752 exploit-URLs would be useful for detecting zero-day exploits because the fact that we could find them quickly within the first month implies that they are more popular and easier to reach. Although zero-day exploits are extremely powerful, they need to connect to popular web sites in order to receive traffic to exploit. If they connect to any of the monitored URLs in our list, the HoneyMonkey can quickly detect the exploits and identify the exploit providers behind the scene through redirection analysis. Our zero-day exploit detection experience confirmed the effectiveness of this approach.

(2) Monitoring content providers with well-known URLs is effective: we predicted that monitoring content providers would be useful for tracking the potentially dynamic behavior of exploit providers. Unlike exploit providers who could easily move from one IP address to another and use random URLs, content providers need to maintain their well-known URLs in order to continue attracting browser traffic. The HoneyMonkey takes advantage of this fundamental weakness in the browser-based exploit model and utilizes the content providers as convenient entry points into the exploit network. Again, our zero-day exploit detection experience confirmed the effectiveness of this approach.

(3) Monitoring highly ranked and advanced exploit-URLs is effective: we predicted that the top exploit sites we identified are more likely to upgrade their exploits because they have a serious investment in this business. Also, web sites that appear in the SP2-PP graph are more likely to upgrade because they appeared to be more up-to-date exploiters. Both predictions have been shown to be true: the first detected zero-day exploit-URL belongs to the #9 site in Figure 5 (which is registered to the same email address that also owns the #4 and #5 sites) and 7 of the top 10 sites in Figure 5 upgraded to the javaprxy.dll exploit; nearly half of the SP2-PP exploit-URLs in Figure 4 upgraded as well.

4.3. Scanning Popular URLs

By specifically searching for potentially malicious web sites, we were able to obtain a list of URLs that have 1.28% of the pages performing exploits. A natural question that most web users will ask is: *if I never visit those risky web sites that serve dangerous or questionable content, do I have to worry about vulnerability exploits?* To answer this question, we gathered the most popular one million URLs as measured by the click-through counts from a search engine and tested them with the HoneyMonkey system. We also compared the results of this popular-list data with the suspicious-list data in Section 4.1. Figure 7 summarizes the comparison of key data.

	Suspicious List	**Popular List**
# URLs scanned	16,190	1,000,000
# Exploit URLs	207 (1.28%)	710 (0.071%)
# Exploit URLs After Redirection (Expansion Factor)	752 (263%)	1,036 (46%)
# Exploit Sites	288	470
SP2-to-SP1 Ratio	204/688 = 0.30	131/980 = 0.13

Figure 7. Comparison of the suspicious-list and popular-list data.

4.3.1. Summary Statistics

Before redirection analysis

Of the one million URLs, HoneyMonkey determined that 710 were exploit pages. This translates into a density of ***0.071%***, which is between one to two orders of magnitude lower than the 1.28% number from the suspicious-list data. The distribution of exploit-URLs among the ranked list is fairly uniform, which implies that the next million URLs likely exhibit a similar distribution and so there are likely many more exploit URLs to be discovered. Eleven of the 710 exploit pages are very popular: they are among the top 10,000 of the one million URLs that we scanned. This demonstrates the need for constant, automatic web patrol of popular pages in order to protect the Internet from large-scale infections.

After redirection analysis:

The Stage-2 HoneyMonkey redirection analysis expanded the list of 710 exploit-URLs to 1,036 URLs hosted by 470 sites. This (1,036-710)/710=*46%* expansion is much lower than the 263% expansion in the suspicious-list data, suggesting that the redirection network behind the former is less complex. The SP2-to-SP1 ratio of 0.13 is lower than its counterpart of 0.30 from the suspicious-list data (see Figure 7). This suggests that overall the exploit capabilities in the popular list are not as advanced as those in the suspicious list, which is consistent with the findings from our manual analysis.

Intersecting the 470 exploit sites with the 288 sites from Section 4.1 yields only 17 sites. These numbers suggest that the degree of overlap between the suspicious list, generally with more powerful attacks, and the popular list is not alarmingly high at this point. But more and more exploit sites from the suspicious list may try to "infiltrate" the popular list to increase their infection base. In total, we have collected ***1,780 exploit-URLs hosted by 741 sites***.

4.3.2. Node ranking

Site ranking based on connection counts

Figure 8 illustrates the top 15 SP1-UP exploit sites by connection counts. There are several interesting differences between the two data sets behind the suspicious-list exploiters (Figure 5) and the popular-list exploiters (Figure 8). First, there is not a single pair of exploit sites in the popular-list data that are doing two-way traffic redirection, which appears to be unique in the malicious pornography community. Second, while it is not uncommon to see web sites redirecting traffic to more than 10 or even 20 sites in the suspicious-list, sites in the popular-list data redirect traffic to at most 4 sites. This suggests that aggressive traffic selling is also a phenomenon unique to the malicious pornography community.

Finally, the top four exploit providers in the popular-list clearly stand out. None of them have any URLs in the original list of one million URLs, but all of them are behind a large number of exploit pages which redirect traffic to them. The #1 site provides exploits to 75 web sites primarily in the following five categories: (1) ***celebrities***, (2) ***song lyrics***, (3) ***wallpapers***, (4) ***video game cheats***, and (5) ***wrestling***. The #2 site receives traffic from 72 web sites, the majority of which are located in one particular country. The #3 site is behind 56 related web sites that serve cartoon-related pornographic content. The #4 site appears to be an advertising company serving exploiting links through web sites that overlap significantly with those covered by the #1 site.

Site ranking based on number of hosted exploit-URLs

Figure 9 illustrates the top 122 sites hosting more than one exploit URL. Unlike Figure 6, which highlights

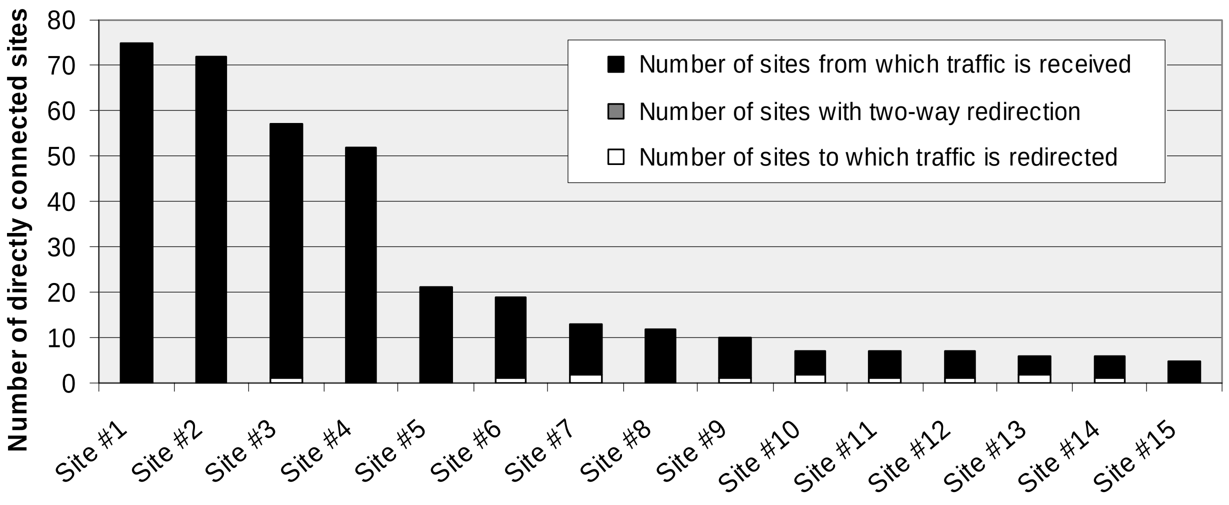

Figure 8. Top 15 exploit sites ranked by connection counts, among the 426 SP1-UP exploit sites

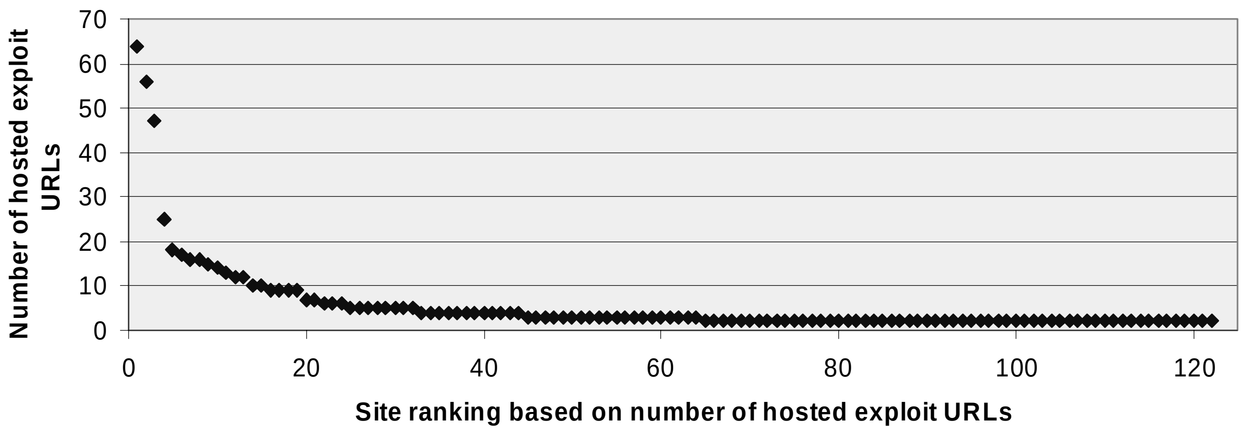

Figure 9. Top 122 sites ranked by the number of exploit-URLs, among the 426 SP1-UP exploit sites

mostly exploit provider sites, Figure 9 highlights many content provider sites that host a large number of exploit pages containing a similar type of content. Again, the top four sites stand out: the #1 site is a content provider of video game cheats information for multiple game consoles. The #2 site (which also appears as the third entry in Figure 8) hosts a separate URL for each different web site from which it receives traffic. The #3 site is a content provider that has a separate entry page for each celebrity figure. The #4 site is a content provider of song lyrics with one entry page per celebrity singer.

5. Discussions

Now that the effectiveness of the HoneyMonkey system is widely known [HM], it is expected that exploit sites will start adopting techniques to evade HoneyMonkey detection. We discuss three types of potential evasion techniques and our countermeasures. Since it has become clear that a weakness of the HoneyMonkey is the time window between a successful exploit that allows foreign code execution and the subsequent execution of the HoneyMonkey exploit detection code, we have developed and integrated a tool called ***Vulnerability-Specific Exploit Detector (VSED)***, which allows the HoneyMonkey to detect and record the first sign of an exploit. Such a detector only works for known vulnerabilities though; detecting zero-day exploits of totally unknown vulnerabilities remains a challenge. The VSED tool will be discussed in Section 5.4.

5.1. Identifying HoneyMonkey Machines

There are three ways for an exploit site to identify HoneyMonkey machines and skip exploits.

(1) Targeting HoneyMonkey IP addresses: The easiest way is to black-list the IP addresses of HoneyMonkey machines. We plan to run the HoneyMonkey network behind multiple ISPs with dynamically assigned IP addresses. If an exploit site wants to black-list all IP addresses belonging to these ISPs, it will need to sacrifice a significant percentage of its infection base. One market research study of ISP client membership [ISP] shows that the top 10 US ISPs service over 62% of US Internet users.

(2) Performing a test to determine if a human is present: Currently, HoneyMonkeys do not click on any dialog box. A malicious web site could introduce a one-time dialog box that asks a simple question; after the user clicks the OK button to prove he's human, the web site drops a cookie to suppress the dialog box for future visits. More sophisticated web sites can replace the simple dialog box with a CAPTCHA Turing Test [ABL04] (although this would raise suspicion because most non-exploiting sites do not use such tests). We will need to incorporate additional intelligence into the HoneyMonkeys to handle dialog boxes and to detect CAPTCHA tests when we see web sites starting to adopt such techniques to evade detection.

(3) Detecting the presence of a VM or the HoneyMonkey code: Malicious code could detect a VM by executing a series of instructions with high virtualization overhead and comparing the elapsed time to some external reference [VMC+05]; by detecting the use of reserved x86 opcodes normally only used by specific VMs [L05]; by leveraging information leaked by sensitive, non-privileged instructions [RP]; and by observing certain file directory contents known to be associated with UML (User-Mode Linux) [CDF+04] or a specific hardware configuration, default MAC address, or I/O backdoor associated with VMware [HR05].

Most VM-detection techniques arise due to the fact that the x86 processors are not fully virtualizable. Fortunately, both Intel [VT] and AMD [PVT] have proposed architecture extensions that would make x86 processors fully virtualizable, and thus make detecting a VM more difficult. In the meantime, we can adopt anti-detection techniques that target known VM-detection methods [CDF+04,VMC+05]. As VMs are increasingly used as general computing platforms, the approach of detecting HoneyMonkeys by detecting VMs will become less effective.

Alternatively, we developed techniques that allow us to also run HoneyMonkey on non-virtual machines so that the results can be cross-checked to identify sophisticated attackers. We implemented support to efficiently checkpoint our system (both memory and disk state) when it is in a known-good state, and roll back to that checkpoint after an attack has been detected. To checkpoint memory, we utilized the hibernation functionality already present in Windows to efficiently store and restore memory snapshots. To support disk checkpoints, we implemented copy-on-write disk functionality by modifying the generic Windows disk class driver which is used by most disks today. Our copy-on-write implementation divides the physical disk into two equally sized partitions. We use the first partition to hold the default disk image that we roll back to when restoring a checkpoint, and the second partition as a scratch partition to store all disk writes made after taking a checkpoint. We maintain a bitmap in memory to record which blocks have been written to so we know which partition contains the most recent version of each individual block. As a result, no extra disk reads or writes are needed to provide copy-on-write functionality and a rollback can be simply accomplished by zeroing out the bitmap. To provide further protection, we can adopt resource-hiding techniques to hide the driver from sophisticated attackers who are trying to detect the driver to identify a HoneyMonkey machine.

Some exploit sites may be able to obtain the "signatures" of the HoneyMonkey logging infrastructure and build a detection mechanism to allow them to disable the logging or tamper with the log. Since such detection code can only be executed after a successful exploit, we can use VSED to detect occurrences of exploits and highlight those that do not have a corresponding file-creation log. Additionally, we are incorporating log signing techniques to detect missing or modified log entries.

We note that some classes of exploits require writing a file to disk and then executing that file for running arbitrary code. These exploits cannot escape our detection by trying to identify a HoneyMonkey machine because our file-based detection actually occurs before they can execute code.

5.2. Exploiting without Triggering HoneyMonkey Detection

Currently, HoneyMonkey cannot detect exploits that do not make any persistent-state changes or make such changes only inside browser sandbox. Even with this limitation, the HoneyMonkey is able to detect most of today's Trojans, backdoors, and spyware programs that rely on significant persistent-state changes to enable automatic restart upon reboot. Again, the VSED tool can help address this limitation.

HoneyMonkeys only wait for a few minutes for each URL. So a possible evasion technique is to delay the exploit. However, such delays reduce the chance of successful infections because real users may close the browser before the exploit happens. We plan to run HoneyMonkeys with random wait times and highlight those exploit pages that exhibit inconsistent behaviors across runs for more in-depth manual analysis.

5.3. Randomizing the Attacks

Exploit sites may try to inject nondeterministic behavior to complicate the HoneyMonkey detection. They may randomly exploit one in every *N* browser visits. We consider this an acceptable trade-off: while this would require multiple scans by the HoneyMonkeys to detect an exploit, it forces the exploit sites to reduce their infection rates by *N* times as well. If a major exploit provider is behind more than *N* monitored content providers, the HoneyMonkey can still detect it through redirection tracking in one round of scans.

Exploit sites may try to randomize URL redirections by selecting a random subset of machines to forward traffic to each time, from a large set of infected machines that are made to host exploit code. Our node ranking algorithm based on connection counts should discourage this because such sites would end up prioritizing themselves higher for investigation. Also, they reveal the identities of infected machines, whose owners can be notified to clean up the machines.

5.4. Vulnerability-Specific Exploit Detector (VSED)

To address some of the limitations discussed above and to provide additional information on the exact vulnerabilities being exploited, we have developed a vulnerability-specific detector, called VSED, and integrated it into the HoneyMonkey. The VSED tool implements a source-code level, vulnerability-specific intrusion detection technique that is similar to IntroVirt [JKD+05]. For each vulnerability, we manually write "predicates" to test the state of the monitored program to determine when an attacker is about to trigger a vulnerability. VSED operates by inserting breakpoints within buggy code to stop execution before potentially malicious code runs, in order to allow secure logging of an exploit alert. For example, VSED would detect a buffer overflow involving the "strcpy" function by setting a breakpoint right before the buggy "strcpy" executes. Once VSED stops the application, the predicate examines the variables passed into "strcpy" to determine if an overflow is going to happen.

To evaluate the effectiveness of VSED for detecting browser-based exploits, we wrote predicates for six recent IE vulnerabilities and tested them against the exploit-URLs from both the suspicious list and the popular list. Although we do not have a comprehensive list of predicates built yet, we can already pinpoint the vulnerabilities exploited by hundreds of exploit-URLs. One limitation of VSED is that it cannot identify zero-day exploits of unknown vulnerabilities.

6. Related Work

There is a rich body of literature on honeypots. Most honeypots are deployed to mimic vulnerable servers waiting for attacks from client machines [H,P04,J04,KGO+05]. In contrast, HoneyMonkeys are deployed to mimic clients drawing attacks from malicious servers.

To our knowledge, there are three other projects related to the concept of client-side honeypots: email quarantine, shadow honeypots, and Honeyclient. Sidiroglou et al. [SK05] described an email quarantine system which intercepts every incoming message, "opens" all suspicious attachments inside instrumented virtual machines, uses behavior-based anomaly detection to flag potentially malicious actions, quarantines flagged emails, and only delivers messages that are deemed safe.

Anagnostakis et al. [ASA+05] proposed the technique of "shadow honeypots" which are applicable to both servers and clients. The key idea is to combine anomaly detection with honeypots by diverting suspicious traffic identified by anomaly detectors to a shadow version of the target application that is instrumented to detect potential attacks and filter out false positives. As a demonstration of client-side protection, the authors deployed their prototype on Mozilla Firefox browsers.

The two client-side honeypots described above are both *passive* in that they are given existing traffic and do not actively solicit traffic. In contrast, HoneyMonkeys are *active* and are responsible for seeking out malicious web sites and drawing attack traffic from them. The former has the advantage of providing effective, focused protection of targeted population. The latter has the advantages of staying out of the application's critical path and achieving a broader coverage, but it does require additional defense against potential traps/black-holes during the recursive redirection analysis. The two approaches are complementary and can be used in conjunction with each other to provide maximum protection.

In parallel with our work, the Honeyclient project [HC] shares the same goal of trying to identify browser-based attacks. However, the project has not published any deployment experience or any data on detected exploit-URLs. There are also several major differences in terms of implementation: Honeyclient is not VM-based, does not use a pipeline of machines with different patch levels, and does not track URL redirections.

Existing honeypot techniques can be categorized using two other criteria: (1) *physical honeypots* [KGO+05] with dedicated physical machines versus *virtual honeypots* built on Virtual Machines [VMW,UML]; (2) *low-interaction honeypots* [P04], which only simulate network protocol stacks of different operating systems, versus *high-interaction honeypots* [J04], which provide an authentic decoy system environment. HoneyMonkeys belong to the category of high-interaction, virtual honeypots.

In contrast with the black-box, state-change-based detection approach used in HoneyMonkey, several papers proposed vulnerability-oriented detection methods, which can be further divided into vulnerability-specific and vulnerability-generic methods. The former includes Shield [WGS+04], a network-level filter designed to detect worms exploiting known vulnerabilities, and IntroVirt [JKD+05], a technique for specifying and monitoring vulnerability-specific predicates at code level. The latter includes system call-based intrusion detection systems [FHS+96,FKF+03], memory layout randomization [ASLR,XKI03], non-executable pages [AA] and pointer encryption [CBJ+03]. An advantage of vulnerability-oriented techniques is the ability to detect an exploit earlier and identify the exact vulnerability being exploited. As discussed in Section 5.4, we have incorporated IntroVirt-style, vulnerability-specific detection capability into the HoneyMonkey.

7. Summary

We have presented the design and implementation of the Strider HoneyMonkey as the first systematic method for automated web patrol to hunt for malicious web sites that exploit browser vulnerabilities. Our analyses of two sets of data showed that the densities of malicious URLs are 1.28% and 0.071%, respectively. In total, we have identified a large community of 741 web sites hosting 1,780 exploit-URLs. We proposed using topology graphs based on redirection traffic to capture the relationship between exploit sites and using site ranking algorithms based on the number of directly connected sites and the number of hosted exploit-URLs to identify major players. Our success in detecting the first-reported, in-the-wild, zero-day exploit-URL of the javaprxy.dll vulnerability provided the best demonstration of the effectiveness of our approach by monitoring easy-to-find content providers with well-known URLs as well as top exploit providers with advanced exploit capabilities. Finally, we discussed several techniques that malicious web sites can adopt to evade HoneyMonkey detection, which motivated us to incorporate an additional vulnerability-specific exploit detection mechanism to complement the HoneyMonkey's core black-box exploit detection approach.

Acknowledgement

We would like to express our sincere thanks to the anonymous reviewers and our shepherd Nick Weaver for their valuable comments.

References

[AA] S. Andersen and V. Abella, "Data Execution Prevention. Changes to Functionality in Microsoft Windows XP Service Pack 2, Part 3: Memory Protection Technologies.," http://www.microsoft.com/technet/prodtechnol/winxppro/maintain/ sp2mempr.mspx.

[ABL04] L. von Ahn, M. Blum, and J. Langford, "Telling Humans and Computers Apart Automatically," *Communications of the ACM*, Feb. 2004.

[AL] Alexa, http://www.alexa.com/.

[ASA+05] K. Anagnostakisy, S. Sidiroglouz, P. Akritidis, K. Xinidis, E. Markatos, and A. Keromytis. "Detecting Targeted Attacks Using Shadow Honeypots," in *Proc. USENIX Security Symposium*, August 2005.

[ASLR] PaX Address Space Layout Randomization (ASLR). http://pax.grsecurity.net/docs/aslr.txt.

[B04] Xpire.info, http://www.vitalsecurity.org/xpire-splitinfinity-serverhack_malwareinstall-condensed.pdf, Nov. 2004.

[CBJ+03] C. Cowan, S. Beattie, J. Johansen, and P. Wagle. "PointGuard: Protecting pointers from buffer overflow vulnerabilities," in *Proc. USENIX Security Symposium*, August 2003.

[CDF+04] C. Carella, J. Dike, N. Fox, and M. Ryan, "UML Extensions for Honeypots in the ISTS Distributed Honeypot Project," in *Proc. IEEE Workshop on Information Assurance*, 2004.

[CWS05] "Webroot: CoolWebSearch Top Spyware Threat," http://www.techweb.com/showArticle.jhtml?articleID=160400314, TechWeb, March 30, 2005.

[D04] Download.Ject, http://www.microsoft.com/security/incident/download_ject.mspx, June 2004.

[E04] Ben Edelman, "Who Profits from Security Holes?", Nov. 2004, http://www.benedelman.org/news/111804-1.html.

[F04] "Follow the Money; or, why does my computer keep getting infested with spyware?" http://www.livejournal.com/users/tacit/125748.html.

[FHS+96] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longsta, "A sense of self for Unix processes," in *Proc. IEEE Symp. on Security and Privacy*, May 1996.

[FKF+03] H. Feng, O. Kolesnikov, P. Fogla, W. Lee, and W. Gong, "Anomaly detection using call stack information," in *Proc. IEEE Symp. on Security and Privacy*, May 2003.

[G05] "Googkle.com installed malware by exploiting browser vulnerabilities," http://www.f-secure.com/v-descs/googkle.shtml.

[G04] Archana Ganapathi, Yi-Min Wang, Ni Lao, and Ji-Rong Wen, "Why PCs Are Fragile and What We Can Do About It: A Study of Windows Registry Problems", in *Proc. IEEE DSN/DCC*, June 2004.

[H] The Honeynet Project, http://www.honeynet.org/.

[HC] Honeyclient Development Project, http://www.honeyclient.org/.

[HD05] "Another round of DNS cache poisoning," Handlers Diary, March 30, 2005, http://isc.sans.org/.

[HF] hpHOSTS community managed hosts file, http://www.hosts-file.net/downloads.html.

[HM] Strider HoneyMonkey Exploit Detection, http://research.microsoft.com/HoneyMonkey.

[HR05] T. Holz and F. Raynal, "Detecting Honeypots and other suspicious environments," in *Proc. IEEE Workshop on Information Assurance and Security*, 2005.

[IF05] "iframeDOLLARS dot biz partnership maliciousness," http://isc.sans.org/diary.php?date=2005-05-23.

[ISP] ISP Ranking by Subscriber, http://www.isp-planet.com/research/rankings/index.html.

[IW05] "Scammers use Symantec, DNS holes to push adware," InfoWorld.com, March 7, 2005, http://www.infoworld.com/article/05/03/07/HNsymantecholesandadware_1.html?DESKTOP%20SECURITY.

[J04] Xuxian Jiang, Dongyan Xu, "Collapsar: A VM-Based Architecture for Network Attack Detention Center", in *Proc. USENIX Security Symposium*, Aug. 2004.

[J105] Microsoft Security Advisory (903144) - A COM Object (Javaprxy.dll) Could Cause Internet Explorer to Unexpectedly Exit, http://www.microsoft.com/technet/security/advisory/903144.mspx.

[J205] Microsoft Security Bulletin MS05-037 - Vulnerability in JView Profiler Could Allow Remote Code Execution (903235), http://www.microsoft.com/technet/security/bulletin/ms05-037.mspx.

[JKD+05] Ashlesha Joshi, Sam King, George Dunlap, Peter Chen, "Detecting Past and Present Intrusions Through Vulnerability-Specific Predicates," in *Proc. SOSP*, 2005.

[KGO+05] Sven Krasser, Julian Grizzard, Henry Owen, and John Levine, "The Use of Honeynets to Increase Computer Network Security and User Awareness", in *Journal of Security Education*, pp. 23-37, vol. 1, no. 2/3. March 2005.

[L05] Lallous, " Detect if your program is running inside a Virtual Machine," March 2005, http://www.codeproject.com/system/VmDetect.asp.

[M52] Microsoft Security Bulletin MS05-002, Vulnerability in Cursor and Icon Format Handling Could Allow Remote Code Execution, http://www.microsoft.com/technet/security/Bulletin/MS05-002.mspx.

[M311] Microsoft Security Bulletin MS03-011, Flaw in Microsoft VM Could Enable System Compromise, http://www.microsoft.com/technet/security/Bulletin/MS03-011.mspx.

[M413] Microsoft Security Bulletin MS04-013, Cumulative Security Update for Outlook Express, http://www.microsoft.com/technet/security/Bulletin/MS04-013.mspx.

[NK04] Neal Krawetz, Anti-honeypot technology, Security & Privacy Magazine, IEEE Volume 2, Issue 1, Jan.-Feb. 2004 Page(s):76–79.

[P04] Niels Provos, "A Virtual Honeypot Framework", in *Proc. USENIX Security Symposium*, Aug. 2004.

[PVT] AMD Pacifica Virtualization Technology, http://enterprise.amd.com/downloadables/Pacifica.ppt.

[R05] "Russians use affiliate model to spread spyware," http://www.itnews.com.au/newsstory.aspx?CIaNID=18926.

[R04] Team Register, "Bofra exploit hits our ad serving supplier," http://www.theregister.co.uk/2004/11/21/register_adserver_attack/, November 2004.

[RP] Red Pill, http://invisiblethings.org/papers/redpill.html.

[S04] Symantec Gateway Security Products DNS Cache Poisoning Vulnerability, http://securityresponse.symantec.com/avcenter/security/Content/2004.06.21.html.

[S05] "Michael Jackson suicide spam leads to Trojan horse," http://www.sophos.com/virusinfo/articles/jackotrojan.html, Sophos, June 9, 2005.

[SH] "What is Strider HoneyMonkey," http://research.microsoft.com/honeymonkey/article.aspx, Aug. 2005.

[SK05] Stelios Sidiroglou and Angelos D. Keromytis, "A Network Worm Vaccine Architecture," in *1st Information Security Practice and Experience Conference (ISPEC)*, April 2005.

[T05] Michael Ligh, "Tri-Mode Browser Exploits - MHTML, ANI, and ByteVerify," http://www.mnin.org/write/2005_trimode.html, April 30, 2005.

[UML] Know Your Enemy: Learning with User-Mode Linux. Building Virutal Honeynets using UML, http://www.honeynet.org/papers/uml/.

[VMC+05] Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex Snoeren, Geoff Voelker, and Stefan Savage, "Scalability, Fidelity and Containment in the Potemkin Virtual Honeyfarm," in *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2005.

[VMW] Know Your Enemy: Learning with VMware. Building Virutal Honeynets using VMware, http://www.honeynet.org/papers/vmware/.

[VT] Vanderpool Technology, Technical report, Intel Corporation, 2005.

[W03] Yi-Min Wang, et al., "STRIDER: A Black-box, State-based Approach to Change and Configuration Management and Support", in *Proc. Usenix LISA*, Oct. 2003.

[W04] Yi-Min Wang, et al., "Gatekeeper: Monitoring Auto-Start Extensibility Points (ASEPs) for Spyware Management", in *Proc. Usenix LISA*, 2004

[W05] Yi-Min Wang, Doug Beck, Binh Vo, Roussi Roussev, and Chad Verbowski, "Detecting Stealth Software with Strider GhostBuster," in *Proc. DSN*, June 2005

[WGS+04] Helen J. Wang, Chuanxiong Guo, Daniel R. Simon, and Alf Zugenmaier, "Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits," in *Proc. ACM SIGCOMM*, August 2004.

[XKI03] J. Xu, Z. Kalbarczyk and R. K. Iyer, "Transparent Runtime Randomization for Security," in *Proc. Symp. Reliable and Distributed Systems (SRDS)*, October 2003.

[XSS] "Code insertion in Blogger comments", March 28, 2005, http://www.securityfocus.com/archive/1/394532.