PhoneyC: A Virtual Client Honeypot

Jose Nazario
jose@monkey.org

April 1, 2009

Abstract

The number of client-side attacks has grown significantly in the past few years, shifting focus away from defendable positions to a broad, poorly defended space filled with vulnerable clients. Just as honeypots enabled deep research into server-side attacks, honeyclients can permit the deep study of client-side attacks. A complement to honeypots, a honeyclient is a tool designed to mimic the behavior of a user-driven network client application, such as a web browser, and be exploited by an attacker's content. These systems are instrumented to discover what happened and how. This paper presents PhoneyC, a honeyclient tool that can provide visibility into new and complex client-side attacks. PhoneyC is a *virtual honeyclient*, meaning it is not a real application but rather an emulated client. By using dynamic analysis, PhoneyC is able to remove the obfuscation from many malicious pages. Furthermore, PhoneyC emulates specific vulnerabilities to pinpoint the attack vector. PhoneyC is a modular framework that enables the study of malicious HTTP pages and understands modern vulnerabilities and attacker techniques.

1 Introduction

Client-side attacks have radically changed the information security landscape in recent years. A significant portion of attackers in the past 20 years have focused on server-side goals and vulnerabilities. With two changes in the past few years, client-side attacks have become more successful for attackers. First, the web browser has become the de-facto tool to access Internet resources and is used for banking, communications, and every day life. To accomplish this, the web client has become more feature-rich and, in parallel, more complex, leading to many security vulnerabilities for attackers to exploit. Secondly, Internet servers have become significantly more hardened in the past 5 years than they were in previous years, significantly raising the bar for attackers to gain access. The client has become the weakest part of a network. This shift in the landscape has necessitated a change in exploit discovery and analysis.

An ever growing number of HTTP client-side attacks have been discovered and launched on the Internet [19]. In this time the complexity and sophistication of attacks has also grown, including obfuscation techniques and encryption, as well as server-side counter-surveillance techniques. Also in this time we have seen the appearance of exploit "packs" designed to facilitate the attacker's activities [13]. These toolkits are able to construct dynamic HTML pages that encode many exploits into a single site in an attempt to infect the host. When coupled to massive website manipulations, these toolkits can infect thousands of PCs.

One way of studying such toolkits is to use a client that is designed to be exploited so that the effects may be studied. We call such systems honeyclients [21], a modification of the honeypot theme. Just as with honeypots, honeyclients can be high interaction to simulate all aspects of the client operating system, whereas a low-interaction honeypot only simulates the client application. We can also differentiate between real and virtual honeyclients. Real honeyclients use the actual application that would be attacked, whereas virtual honeyclients emulate the application in software. In this paper we refer to PhoneyC as a low interaction, virtual honeyclient because it only emulates the core functionality of a web client and no underlying OS features.

Virtual honeyclients are an attractive tool to use to study such content because they do not require additional computers to analyze vast amounts of malicious content and can easily scale. Using virtual honeyclients it is possible to inspect many websites in parallel with only one real system. However, to convince a website that it is talking to a legitimate client application we need to mimic the application's behavior and responses to inspection.

This paper provides a brief overview of the design and implementation of PhoneyC, a client program that emulates a fully featured HTTP client. PhoneyC supports various client emulations, dynamic languages such as JavaScript, and mimics ActiveX add-ons, as well. PhoneyC can process a suspect web page and analyze the script bodies and react to the dynamic portions of the website. PhoneyC can be configured to mimic a variety of common web clients.

To analyze the malicious content, obfuscated or encrypted JavaScript is decoded and reanalyzed, mimicking

what the real web browser would do with such content. PhoneyC is the first virtual honeyclient that can perform dynamic analysis of JavaScript and Visual Basic Script to remove obfuscation. PhoneyC's virtual features are instrumented to understand their vulnerable functions and to provide alerts. Furthermore, downloaded content is scanned using an antivirus engine to look for known malicious content. PhoneyC can walk a malicious website and report on the exploit chain. We also provide an experimental evaluation of PhoneyC that shows that it can successfully decode and analyze exploit websites found in the wild.

The rest of this paper is organized as follows. Section 2 gives background information on PhoneyC's design and implementation. In section 3 we present an evaluation of PhoneyC in which we demonstrate its performance against real exploit sites found in the wild. Limitations of PhoneyC and virtual honeyclients are described in section 4. In section 5 we discuss the limitations of the current PhoneyC design and discuss how it may be improved in the future. We provide related work in section 6 and conclude in section 7.

2 Design and Implementation

A virtual HTTP honeyclient is a combination of a web crawler and an analysis engine. Like a web crawler, it must be able to evaluate a web page and determine the links that lead out from the page. However, unlike a standard web crawler, it must be able to analyze the page to determine if it is malicious or benign.

PhoneyC is implemented in the Python language [20] to aid in rapid development and extendibility, as well as integration with other tools and libraries. Python was also chosen to minimize security flaws that may appear in an implementation such as C or C++. At its core, PhoneyC has two major components: an input collector and an input evaluator. The input collector is simply a call to the Curl tool [23] together with arguments to mimic a legitimate browser's behavior.

The basic requirements for PhoneyC are:

- PhoneyC must be able to convince a website that it is a legitimate web browser to collect the content that would be sent to an actual user.
- Second, PhoneyC must be able to enumerate all of the links from the HTML page and visit them. This includes all common HTML tags as well as generated HTML content, IFRAMEs, and redirections.
- Third, PhoneyC must be able to understand and evaluate dynamic content such as JavaScript and Visual Basic Script. Fourth, PhoneyC must be able to detect malicious content and provide it for further analysis.
- Finally, because PhoneyC is analyzing hostile code, it must be resilient to attacks itself.

Basic data flow through PhoneyC is shown in Figure 1. URLs are fed to the system to initiate the evaluation, together with a referring URL when available (e.g. when a link is followed). The client retrieves the data using Curl from the server and stores all of the content on disk for additional analysis, if needed, and the data is scanned using an antivirus engine (ClamAV [8]). Curl, running as a subprocess, provides a robust HTTP client in a separate process space, effectively as a sandbox to minimize client system attacks.

If the content is HTML, the SGML parser collects the script content for JavaScript and Visual Basic Script (VBS). No restructuring of the input HTML (such as DOM validation or tag balancing) is performed which may alter the exploit. The parser also collects all of the outgoing links and normalizes them as needed. Links can be present as the common A and IMG tags, as well as IFRAME tags and redirects. The script bodies are analyzed as described below. The parsed page is stored as an object with all of its contents and attributes as a page object referenced by the origin URL. These attributes include the complete script bodies and all outgoing links. PhoneyC also understands major page events such as "onLoad()", but does not handle form submission or emulate mouse clicks. Once the full page has been analyzed, PhoneyC repeats the process for the next page with an outgoing link and the referring URL as inputs.

PhoneyC is provided as a honeyclient module and an HTTP client script, "honeywalk", which calls the core methods of the honeyclient module. Honeywalk is demonstrated below. Other HTTP honeyclient tools can be build on the existing honeyclient module.

PhoneyC is designed to crawl websites to discover exploits and also to act as a malcode collector. It is not, however, designed to extensively crawl the entire world wide web. As such, the scalability and speed deficiencies present in PhoneyC are not a pressing issue at this time.

2.1 Anti-Analysis Techniques

With the growth of web exploits has come a growth in their analysis. To bypass such analysis, attackers have begun applying obfuscation and encryption to their web exploit pages. This can make static analysis more challenging, but not impossible. In the past year we have seen more anti-analysis techniques appear that are designed to defeat illegitimate script access by tools such as PhoneyC.

One of the major goals of PhoneyC was to ease the analysis of complex malicious websites. To do this, PhoneyC must be able to mimic a legitimate web browser to the

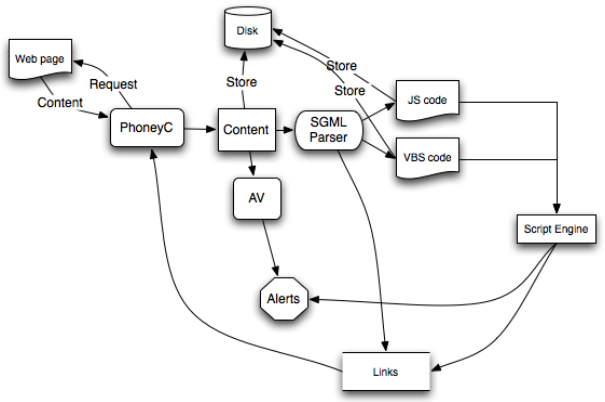

Figure 1: Data flow within PhoneyC. One or more URLs are used to feed the client, which retrieves the content from the server. This is then stored on disk, scanned by AV for any suspect content, and also passed to an SGML parser for evaluation if it is HTML. The SGML parser breaks out script code by language for analysis by the specific script engine. The SGML parser also collects and normalizes any of the links from the HTML page as well as the output of the script output. Foreign scripts not included in the page, using the "src" argument to the SCRIPT tag, may also be collected and analyzed in the context of the current page. The script engines provide alerts to the system, as well.

server to receive the proper content, and to properly interpret the content to decode it. PhoneyC has been able to keep pace with most changes in the malicious website threat landscape, although some changes have been made during development.

While some sites will send all exploit pages to all visitors, many sites will selectively direct clients to specific pages on the basis of their browser software. The following JavaScript differentiates between Microsoft Internet Explorer and other browsers, incorporating an IFRAME element with Internet Explorer-specific exploits if MSIE is found in the User-agent header.

```
if(navigator.userAgent.toLowerCase().indexOf("msie")>0)
  document.write("<iframe src=fl/ifl.html width=100
                  height=0>");
else
  {document.write("<iframe src=fl/ffl.html width=100
                   height=0>")}
```

Obviously more sophisticated techniques to differentiate between browsers is possible. The User-agent header check may also be done on the server using, for example, a PHP script.

To thwart such counter-analysis techniques, PhoneyC attempts to mimic a legitimate web surfer using a legitimate web browser. "Personalities" are created through the use of the User-agent header [2] to mimic Internet Explorer 6 on Windows XP (the default) or Mozilla Firefox 2 on Windows XP. The JavaScript "navigator" object should also correctly mimic the client browser version. Other browsers may be mimicked through additional header forgery and script responses. Furthermore, if possible referring URLs are sent to the web server in the HTTP client headers, another check that some sites perform to ensure that only legitimate victims are served malicious content.

Once the content is presented by the server, additional measures may be in place to prevent detection by an external network-based tool such as an IDS or to slow down an analyst's inspection of the content. These sorts of methods often include variable name obfuscation or base-64 encoding of the content. More complicated techniques seen in the wild to thwart analysis may include:

- Regular expression-based substitutions or removal of junk characters.
- Compression of the script code, using an openly available script compressor that is popular in the web development crowd.
- Encoding of the script code to provide some limited, basic encryption. We have seen more sophisticated sites use the page's URL as an encryption key, so that if the page is shared without the true URL it cannot be decrypted easily.
- True encryption of the code using an encryption algorithm such as RSA.

Additional methods have been seen and may be layered or repeatedly applied. Multi-stage encodings are not uncommon. The browser decodes one part of the page and uses the output of that part as a component of the script for the next part, such as a decoder routine. Some of these techniques have been adopted by attackers from existing, benign JavaScript protection tools and compressors, designed to optimize page load times [7].

2.2 Script Parsing

Dynamic content in the form of JavaScript and Visual Basic Script is executed in a limited environment to perform dynamic analysis. JavaScript processing is done in a subprocess using the SpiderMonkey interpreter [4]. The web page's script body is collected and aggregated for analysis. A basic environment is prepended to the script body to mimic the browser's features, including the *document* object as well as *window* and the *navigator* object. Basic DOM inspection features, such as *getElementById* and others are implemented as well.

Script obfuscation, very common in web page exploits [13], is bypassed with the script interpreter and some basic overrides in the script preamble. The "eval()" method,

common to execute a newly decoded block of code, is overridden with a modified version that can recover if an error is observed by rerunning the script code with any of the output of previous runs. This effectively bypasses multi-stage decoders or decrypters where new code (e.g. a decrypter engine) is written to the page and then used in the next script interpretation. The modified version of the "eval()" method then calls the real "eval()" method to get the proper result and prepends it to the script body as needed.

Visual Basic Script (VBS) code is analyzed using the vb2py package [12] which translated VBS scripts into equivalent Python scripts. These scripts are then analyzed using a child Python interpreter process and the output is collected. The VBS subsystem is not yet as well developed as the JavaScript subsystem, but is designed to accomplish the same results.

2.3 Vulnerability Modules

Detecting specific vulnerabilities to both classify them and to respond to them is a key design goal of PhoneyC. Rather than relying on external patterns or anti-virus alone which may not completely detect malicious HTML content, PhoneyC performs dynamic analysis of the content to determine the vulnerability and analyze the next action.

PhoneyC uses vulnerability modules to mimic vulnerable HTTP client extensions, including ActiveX controls and core browser functionality. These are similar to vulnerability modules in the MWCollect virtual honeypot [5] in that they are vulnerability-specific. However, unlike the MWCollect modules, they do not rely on matching shellcodes or patterns. Instead, these modules look for exploit activity against a vulnerable method independent of the payloads. In a real browser, these objects are dynamically created in the script code and provide an interface between the browser and system libraries. In PhoneyC, these objects create pure JavaScript or VBS objects that implement core functional methods that are exploited.

By using a virtual honeyclient and the vulnerability module architecture, PhoneyC avoids a number of real-world constraints. The primary challenge to discovering exploits on web pages with a real honeyclient is creating a vulnerable system. Often the necessary add-ons are not present. Another issue is the issue of language packs. In many cases an exploit is designed to work specifically on one Windows language pack but will not work on others. Virtual honeyclients can more quickly load more vulnerable modules than a real honeyclient, and they can analyze the exploit code for multiple language packs more easily.

An example vulnerability module, implementing checks for the WebViewFolderIcon.setSlice() attack (CVE reference ID CVE-2006-3730) is shown below. In this vulnerability, a magic value of 0x7ffffffe passed as the first argument to the "setSlice()" method enables arbitrary code execution by the attacker [10].

```
function WebViewFolderIcon() {
    this.setSlice=function(arg0, arg1, arg2, arg3) {
        if (arg0 == 0x7ffffffe) {     // magic value
            add_alert('WebViewFolderIcon.setSlice attack');
        }
    }
}
```

This JavaScript is one of many modules prepended to the page's JavaScript code that is analyzed by the system for any web page that includes script code. In this case the code looks for a magic value in the first argument to the setSlice() method and alerts when such code is found. Similar modules for other vulnerabilities can look for overly long arguments or arbitrary file access.

Creating new modules is relatively easy and requires two pieces, the vulnerability module and a reference to it via the ActiveX CLSID values. The module code itself simply creates a class and the appropriate methods and arguments together with argument inspection to determine when to alert. Furthermore, the modules can be written in the complete absence of any exploit code. All that is needed is to know the basics of the vulnerability, such as the method names and the nature of the malicious arguments, such as "an argument longer than 400 bytes leads to a stack overflow". Based on those conditions a simple argument scanner can be developed. The class is then referenced in the ActiveX map by the CLSID, both by hexadecimal values and by a simplified name. This provides a new HTML *object* is created, mapping the object ID and the CLSID to the right script class.

Such an example is the NctAudioFile2 ActiveX control buffer overflow. A vulnerability description was used to create the vulnerability module, with analysis that reads in part [15]:

> The vulnerability is caused due to a boundary error in the NCTAudioFile2.AudioFile ActiveX control when handling the "SetFormatLikeSample()" method. This can be exploited to cause a stack-based buffer overflow by passing an overly long string (about 4124 bytes) as argument to the affected method.

Based on this description, the following vulnerability module was created:

```
function NCTAudioFile2() {
   this.SetFormatLikeSample=function(arg) {
      if (arg.length > 4124) {
         add_alert('NCTAudioFile2 overflow in
                    SetFormatLikeSample');
      }
   }
}
```

Similar vulnerability modules can be written for exploits that use malicious object properties through the JavaScript

"watch()" method, that handles property changes. The callback function to watch performs similar argument inspection to alert if an exploit scenario is encountered.

When an unknown CLSID is found, one for which PhoneyC has no modules, a message is generated. This can be used to develop new modules which may indicate their use in exploits.

At this time over 65 unique vulnerability modules exist and are usable by PhoneyC. Major vulnerability modules include handlers for Yahoo Messenger, RealPlayer, and the WebFolderViewIcon handler in Internet Explorer 6. New modules are frequently added based on vulnerability reports and exploit code.

3 Evaluation

For a medium-scale evaluation of PhoneyC 470 unique URLs were gathered that were suspected drive-by download sites and web browser exploits. These URLs were gathered from various sites and materials including the URL blacklist maintained at lineage.paix.jp, the drive-by malware analysis blog at ilion.blog47.fc2.com, and the author's own suspect URL collection based on spam traps. URLs were submitted to the MITRE honeyclient system run by Honeyclient.org for analysis and also run through PhoneyC. The test system for PhoneyC was a MacBook running OS X 10.4.11 on an Intel Core Duo with a 2GHz clock speed and 2GB of SDRAM using Python 2.3.5 and SpiderMonkey 1.6. To speed up performance 24 PhoneyC processes were run in parallel. Each PhoneyC process was allowed a maximum depth of 4 links from the root URL.

Of the 470 candidate URLs, 115 were live (yielding a 200 OK), 14 URLs yielded a 300-series redirect, 42 yielded an HTTP return code of 400 (a Bad request), 21 yielded a 401 (unauthorized) error, 149 yielded an HTTP 404 error (the URI was not found on the server), 269 were unreachable (the host was not responding), and the remaining sites' DNS names were unable to resolve. In total about 1.1MB of HTML text was downloaded and analyzed.

3.1 Performance

PhoneyC's performance is hampered by design weaknesses, including calling out to external processes such as Curl, ClamAV, and SpiderMonkey to assist with the analysis, as well as only working on one URL at a time. The 470 unique URLs analyzed in this evaluation data set provide a representative sample of PhoneyC's abilities. Some URLs were analyzed faster than others. URL evaluation took between 3 seconds at a minimum and over 3.2 hours at a maximum for any specific URL, depending on its complexity (number of outbound links and any encountered script evaluation time). The average URL took 2.1 hours to analyze all outbound links and script code at a maximum distance of 4 references, including images as well as links off of the page. Note that PhoneyC can be slowed down dramatically if the HTTP server for the URL is unreachable.

In contrast, the MITRE honeyclient, in contrast, was able analyze the 470 URLs in approximately 14 minutes. This is due to using a native browser, working in even greater parallel, and only loading the one URL (including images and scripts) before leaving the URL.

3.2 Accuracy and Insights

From the 470 URLs, 22 unique HTML pages (unique via MD5 checksums) were downloaded, of which 2 yielded hits in ClamAV (both for the signature Exploit.CVE-2006-3730). Other scanners tested included Kaspersky Antivirus, BitDefender, Grisoft's AVG, and Fortinet's 'vscanner' tool, none of which yielded signature hits on the HTML downloaded by PhoneyC. Over 2700 script bodies were evaluated (through a series of dynamic evaluations by PhoneyC), only three of which yielded positive signature hits with ClamAV for JS.Dropper-33 and Exploit.HTML.IFrameBOF-4.

Dynamic analysis by PhoneyC revealed that the most popular exploits found in the URL corpus were for the Xunlei Thunder 5.x DownURL2() overflow [17] (6 found in total) and the PPStream (PowerPlayer.dll 2.0.1.3829) ActiveX Remote Overflow Exploit in the Logo property [3] (6 found in total in this data set). PhoneyC is able to emulate this ActiveX control that has been widely exploited as a malcode dropper. As described below, PhoneyC's limits means that it is unable to capture all exploits seen in the wild, however.

In contrast to PhoneyC's findings, the MITRE honeyclient setup found only 4 URLs that registered an alert. These differences highlight weaknesses of PhoneyC as well as some of its strengths.

URLs that registered an alert in PhoneyC did not always register issues in the MITRE honeyclient setup. An example is the URL http://www.pineapple4u.com/relogonn.htm. This was flagged as containing an exploit in PhoneyC (Exploit.CVE-2006-3730) even though the JavaScript subsystem was unable to process the script correctly. This is due to the combined approach used by the tool to analyze URLs.

Another example is the URL http://www.lineagecojp.com/tmsn/StormII.htm from the feed of URLs processed by both PhoneyC and the MITRE honeyclient. In this case PhoneyC's dynamic script analyzer found an issue with the page and flagged an exploit for the PPStream (PowerPlayer.dll 2.0.1.3829) ActiveX Overflow in the rawParse() method, even though ClamAV did not register an issue. The MITRE honeyclient did not flag this host.

Blacklist domain	Number of domains blacklisted
sophosxl.com	83
surbl.org	102
mailshell.net	90

Table 1: Number of URLs screened by PhoneyC blacklisted by major DNS-based blacklists. This indicates that these URLs are possibly suspicious based on the domain name status.

An example where both PhoneyC and the MITRE honeyclient agreed that a URL was malicious was with the root exploit URL http://sexbases.cn/in.cgi?16&1dfcb2. This URL was flagged as suspicious by the MIYRE honeyclient. PhoneyC is able to reveal that through a series of scripts and IFRAMEs the real exploit URL is http://dasretokfin.com/index.php, which exploits Exploit.CVE-2006-3730.

PhoneyC missed some of the URLs flagged by the MITRE honeyclient, such as http://www.lineagecojp.com/ie/Ms06014.htm. In this case this is a commonly found exploit for an XML Request object in Internet Exploit 6. This emulation is not fully available in PhoneyC at this time, so the dynamic script analyzer was unable to correctly analyze this URL. Another URL missed by PhoneyC was http://freeonlinehostguide.com/index.php. The MITRE honeyclient correctly recognized an exploit on this page but PhoneyC was not able to fetch any HTML contents for this URL and so was not able to determine if the site was malicious. This is commonly due to poor emulation of the Internet Explorer request to a web server which looks for specific browser request features before serving the content.

Hostname blacklists were also used to compare the findings and any suspicious nature of the URLs. If a domain name for the URL appears on a blacklist the URL itself may be malicious. To test this, and to verify that the URLs were possibly malicious, three blacklists were queried for the hostnames in the URLs. The results of any blacklisting are shown in Table 1.

4 PhoneyC Detection

Virtual honeyclients are open to a number of detection techniques and attack vectors. A number of these issues are present in any virtual tool due to the limits of software emulation and will be present in any virtual honeyclient.

PhoneyC specifically is vulnerable to detection through a simple check for the "page_alerts" array in any of the JavaScript code. This variable name is not hidden at all and is accessible by any of the script code. A malicious page can simple check for the existence of this variable by reference and exit if it is found. To remedy this, the variable reference can be randomly generated, for instance.

As shown in the previous section, PhoneyC is also much slower than a normal web browser, enabling timing attacks to be used against it. This is an extension of a simple debugger check which uses the time elapsed between two set points to detect single stepping. A malicious server that measures the time between requests that should be very short can deny the client content. Performance improvements and concurrent requests to mimic a standard web browser would remedy this issue.

When impersonating Internet Explorer, PhoneyC can be detected through the SpiderMonkey JavaScript interpreter, which differs slightly from the script engine in the real Internet Explorer. Certain behaviors are well known and derive from ambiguities in the JavaScript specification, such as regular expression handling. Any virtual honeyclient that relies on SpiderMonkey will suffer the same issues.

Finally, any virtual honeyclient will always fail to emulate all aspects of a real browser. Dynamic content can be used to inspect arbitrary features of the browser using calls that may not be documented. As such, suspicious sites may be able to detect virtual honeyclients by calling methods that are not implemented. This parallels attacks to detect virtual execution environments used in malware sandboxing [11] and is a fundamental flaw of any kind of emulation.

5 Future Work

PhoneyC is far from complete at this point, although it is one of the tools the author uses to analyze malicious websites. Currently a number of factors are being evaluated to redesign PhoneyC to add functionality and improve reliability and performance. Minor improvements include easier configuration, proxy support, and performance enhancements. PhoneyC's vulnerability module architecture is being re-evaluated to handle previously unknown ActiveX CLSIDs and methods using a generic approach. Additionally, a generic scripting language framework is being developed to avoid having to write two vulnerability modules for any specific attack vector, one in JavaScript and one in VBS. Major PhoneyC deficiencies and improvements we are evaluating are listed below.

5.1 Exploit Enumeration

One major drawback to the current design of the vulnerability modules is that they can only alert for a single exploit that the system knows of. All other vulnerabilities that may be present in the web page are not analyzed and not

reported. The author previously developed a version of the script analysis engine used in PhoneyC, dubbed "Norberto", that performed static analysis of the page to enumerate multiple exploits in the page after some basic dynamic analysis to decode the page. In the future, PhoneyC may be extended with a similar static analysis engine to enumerate the complete list of exploits in the page.

5.2 Additional Content Types

Since PhoneyC was first developed, a number of new content types have become the focus of exploit activity, including Adobe's Portable Document Format (PDF) and their Flash format. These content types are not understood by the tool and may contain malicious content. Currently, we use manual techniques to analyze malicious Flash and PDF documents. Their increased use on the web as a means to deliver malicious executables to the end user means that we should incorporate their analysis into PhoneyC.

5.3 Shellcode Analysis

PhoneyC is not able to understand shellcode that may be presented in the dynamic HTML page and is currently limited to a generic alert for a vulnerable method. Because of this, an analyst must still perform follow-up analysis of the content to determine the next stage of the attack. We have been working on integrating a shellcode analysis engine, libEmu [1], into the dynamic payload inspection engine to further understand the next stage of the attack. This will also require tighter integration with the SpiderMonkey script engine.

6 Related Work

Previous work at characterizing widespread web-based malware has been described by Provos *et al* [14]. In their work, "drive by" websites were found to be widespread and arising from website compromises and third-party script abuse, such as JavaScript-based visitor counters.

PhoneyC is not the first such honeyclient tool and builds on a number of previous works. The vulnerability module concept was inspired by the MWCollect virtual honeypot daemon [5]. In PhoneyC, modules are designed to mimic vulnerable ActiveX controls and implement the methods in JavaScript. Argument checks validate the input and provide a simple alerting mechanism.

Seifert's HoneyC is a low interaction, virtual honeyclient tool [18]. However, it suffers from a lack of ability to analyze obfuscated dynamic HTML and reliance on Snort signatures for detection, which is easily evaded.

Real honeyclients have been developed by multiple independent groups. Seifert's Capture-HPC is a high interaction honeyclient [9] that can be used with, for example, Windows XP and Internet Explorer to analyze malicious websites. The MITRE HoneyClient uses Internet Explorer running on Windows [21]. Fed a list of URLs, the system will visit the URL if needed or pull the results from a local cache. When the URL is visited, if unexpected system changes occur, the URL is marked as suspicious and the new files are made available for additional analysis. The HoneyMonkey project is another large-scale, automated web crawling effort to discover malicious website [22]. URLs are visited both by an older version of the browser and by a newer, up-to-date version to discover previously unknown and unpatched issues. Both are limited in their attack visibility as they require vulnerable modules to be installed in the client.

Not all honeyclient tools are restricted to HTTP content. The SHIELA honeyclient uses Outlook Express driven by external scripts to discover mail-based threats [16]. Trigger conditions are very similar to those of the MITRE Honeyclient, namely an alert happens if one of a set of illegal operations occurs, such as Windows registry changes, file creation, or specific network traffic.

Honeyclient research is also being done using browser plug-ins. Efforts are also underway to incorporate some of the basic heuristics and detection capabilities of honeyclients into real browsers used by analysts [6].

7 Conclusion

PhoneyC is a virtual honeyclient for analyzing websites. It mimics legitimate web browsers and can understand dynamic content, and is the first virtual honeyclient that can de-obfuscate malicious content for detection. By using vulnerability modules, specific attacks can be pinpointed and characterized much faster than with manual analysis.

We gave an overview of PhoneyC's design and architecture and showed how PhoneyC's analysis engine understands malicious websites. Our experimental evaluation showed that PhoneyC works on current, in the wild malicious websites, albeit with room for improvements in both performance and features. We demonstrated that PhoneyC can determine what malicious software may be loaded onto a system if exploits exist.

PhoneyC is publicly available as source code under a GNU Public License. The subversion repository is available at http://code.google.com/p/phoneyc .

Acknowledgments

I would like to thank Georg Wicherski for his review of this paper, his contributions to the PhoneyC codebase and his assistance in a re-design of the toolkit. I would also like

to thank Marco Cova for helpful discussions and bugfixes with PhoneyC. Chris Lee, Adrian Wiesmann, David Watson, and Christian Siefert all provided a generous review of this manuscript.

References

[1] P. Baecher and M. Koetter. libemu - x86 shellcode detection and emulation, 2007. http://www.mwcollect.org/.

[2] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol. *Work in progress of the HTTP working group of the IETF,¡ URL: ftp://nic. merit. edu/documents/internet-drafts/draft-fielding-http-spec-00. txt*.

[3] dummy. PPStream (PowerPlayer.dll 2.0.1.3829) Activex Remote Overflow Exploit, 2007. http://www.milw0rm.com/exploits/4348.

[4] M. Foundation. SpiderMonkey (JavaScript-C) engine. http://www.mozilla.org/js/spidermonkey/.

[5] F. Freiling, T. Holz, and G. Wicherski. Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks. *LECTURE NOTES IN COMPUTER SCIENCE*, 3679:319, 2005.

[6] O. Hallaraker and G. Vigna. Detecting Malicious JavaScript Code in Mozilla. In *Engineering of Complex Computer Systems, 2005. ICECCS 2005. Proceedings. 10th IEEE International Conference on*, pages 85–94, 2005.

[7] D. Jackson. The Packer 2.0 Threat, 2008. http://www.secureworks.com/research/threats/thepacker/.

[8] T. Kojm. ClamAV homepage. http://clamav.net/.

[9] F. Mara, Y. Tang, R. Steenson, and C. Seifert. Capture-Honeypot Client, 2006.

[10] H. D. Moore. MS Internet Explorer WebViewFolderIcon setSlice() (Multiple Exploits), 2006. http://www.securiteam.com/exploits/6A0060AH5G.html.

[11] K. Natvig. Emulation: how low will you go... *2nd International CARO Workshop: Packers, Decryptors and Obfuscators*, 2008.

[12] P. Paterson. vb2py homepage, 2009. http://vb2py.sourceforge.net/.

[13] N. Provos, P. Mavrommatis, M. Rajab, and F. Monrose. All Your iFrames Point To Us. Technical report, Technical Report provos-2008a, Google Inc, 2008. http://research.google.com/archive/provos-2008a.pdf, 2008.

[14] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu. The ghost in the browser analysis of web-based malware. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 4–4, Berkeley, CA, USA, 2007. USENIX Association.

[15] S. Research. Cool Audio Products NCTAudioFile2 ActiveX Control Buffer Overflow, 2007. http://secunia.com/secunia_research/2007-34/.

[16] J. Rocaspana. SHELIA: A Client HoneyPot For Client-Side Attack Detection, 2009. http://www.cs.vu.nl/~herbertb/misc/shelia/.

[17] Secunia. Xunlei Thunder DapPlayer ActiveX Control Buffer Overflow, 2007. http://secunia.com/advisories/26964/.

[18] C. Seifert, I. Welch, and P. Komisarczuk. HoneyC-The Low-Interaction Client Honeypot. *NZCSRCS,(Hamilton, 2007), Available from http://www.mcs.vuw.ac.nz/cseifert/blog/images/seifert-honeyc.pdf; accessed on*, 10, 2006.

[19] M. Servers. Know Your Enemy: Malicious Web Servers.

[20] G. van Rossum et al. Python Language Website, 2009. http://www.python.org.

[21] K. WANG. Using honeyclients to detect new attacks [C/OL]. *RECON Conference*, 2005.

[22] Y. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King. Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities. In *Proc. NDSS*, 2006.

[23] S. Ward and M. Hostetter. Curl: a language for web content. *International Journal of Web Engineering and Technology*, 1(1):41–62, 2003.