Throwing a MonkeyWrench into Web Attackers Plans

Armin Büscher[1,2], Michael Meier[1], and Ralf Benzmüller[2]

[1] Technische Universität Dortmund, Computer Science VI, Germany
{armin.buescher,michael.meier}@tu-dortmund.de
[2] G Data Software AG Bochum, Security Labs, Germany
{armin.buescher,ralf.benzmueller}@gdata.de

Abstract. Client-based attacks on internet users with malicious web pages represent a serious and rising threat. Internet Browsers with enabled active content technologies such as JavaScript are vulnerable to so-called drive-by downloads. Drive-by downloads are able to automatically infect a victim's system during a single visit of a crafted web page testing various vulnerabilities and installing e.g. malware files or illegal content without user interaction. In this paper we present MonkeyWrench, a low-interaction web-honeyclient allowing automatic identification of malicious web pages by performing static analysis of the HTML-objects in a web page as well as dynamic analysis of scripts by execution in an emulated browser environment. Using this hybrid approach MonkeyWrench overcomes shortcomings of existing low-interaction web-honeyclients in dealing with obfuscated JavaScript while outperforming high-interaction systems. Further MonkeyWrench is able to identify the exact vulnerability triggered by a malicious page and to extract payloads from within obfuscated scripts which are valuable information to security analysts and researchers. Results of an examination of several hundred thousand web pages demonstrate MonkeyWrench's ability to expose rising threats of the web, and to collect malware and JavaScript exploit samples.

Keywords: Web Security, Drive-by downloads, Active content-based attacks, Honeyclients.

1 Introduction

Recently the browser has gained popularity as an infection vector for the installation of malicious files. In contrast to attacks on vulnerable services the attacker does not have to find and scan remote systems. However the challenge of a successful web attack lies in attracting internet surfing users to visit the manipulated contents. So-called drive-by download attacks are able to automatically infect a victim's system during a single visit of a crafted web page testing various browser vulnerabilities and installing e.g. malware files or illegal content without user interaction. Figure 1 depicts the workflow of a typical web-based attack consisting of the following steps: (1) the attacker sets up a distribution site serving malicious content and malicious binaries (f.e. a trojan horse) that the attacker intends to spread; (2) the attacker compromises a web server and injects code in web pages integrating malicious contents from the distribution site or the attacker sets up a website with contents tempting specific users to visit; (3) the victim visits a manipulated “bait” page; (4) the bait page

B. De Decker and I. Schaumüller-Bichl (Eds.): CMS 2010, LNCS 6109, pp. 28–39, 2010.

redirects the victim's browser through one or more steps to contents served by the distribution site; (5) the victim's browser requests the remote contents from the distribution site; (6) the distribution site replies with malicious content that is intended to exploit weaknesses of the victim's browser and/or installed plug-ins.

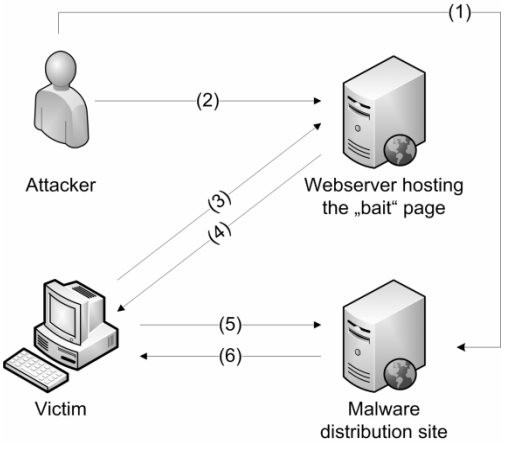

Fig. 1. Workflow of a web-based attack

Traditional security measures like packet filters and NAT do not provide protection against drive-by downloads since the connection is established by the client and allegedly does not pose as a threat. It is a complex and difficult task to establish suitable rules in intrusion detection systems to reliably discover web-based attacks since pattern matching is not applicable to detect obfuscated JavaScript that decrypts itself on the client system during runtime. Many popular websites rely on the clientside execution of script code to provide dynamic contents with enhanced features. Therefore the majority of internet users adopt the default settings of popular browsers to automatically permit JavaScript on every web page. Furthermore a lot of users surf the internet using systems with out-dated and unpatched browsers, operating systems or plug-in-technologies that enable attackers to exploit known vulnerabilities.

Web-honeyclients are used to detect the exploitation of browser vulnerabilities like for drive-by downloads of malware or phishing attacks. Thus the prevalence of web-based attacks can be evaluated and counter measures may be developed. Honeyclients are differentiated into high- and low-interaction systems. Whereas high-interaction systems represent a conventional computer system with real vulnerabilities, low-interaction systems emulate the behavior of a system (cf. [9]). High-interaction web-honeyclients monitor the file system, the system registry and the running processes for changes during a web page visit to heuristically detect malicious behavior. Since the systems are actually vulnerable, the high-interaction honeyclients primarily use virtualized operating systems in order to be easily reset to a predefined state. Resetting the system after each detection obviously limits its availability and performance. Systems following the high-interaction approach reliably detect attacks utilizing obfuscated JavaScript and other active content but need to be protected against malware trying to spread itself and due to the heuristics detection approach offer only a very limited view on the techniques used to obfuscate and carry out web-based attacks. Low-interaction

web-honeyclients check web pages by emulating the behavior of a web browser. Pages are typically checked against a predefined malicious behavior by statically analyzing the HTML-code and performing signature-based detection. Low-interaction web-honeyclients have an advantage in performance over their high-interaction counterparts due to their minor complexity and do not need to be reset after each detection. They however are typically unable to handle obfuscated JavaScript and are therefore prone to most web-based attacks including drive-by downloads.

To overcome the shortcomings of existing low-interaction web-honeyclients, MonkeyWrench follows a hybrid approach that executes and dynamically analyzes the behavior of active content to detect drive-by downloads and provide an analysis of the techniques used to obfuscate attacks and exploit vulnerabilities in a target system. The system has a lower computational overhead than high-interaction web-honeyclients and efficiently solves the problem of obfuscated JavaScript. High-interaction systems have issues detecting malicious sites carrying out attacks against certain combinations of browser version or browser plug-ins. Actually high-interaction architectures have to provide a vulnerable system for each of these combinations whereas MonkeyWrench is able to emulate multiple combinations by repeatedly executing JavaScript in adjusted browser environments when it detects scripts that request properties used to fingerprint the client system. Unlike high-interaction systems that only detect an anomalous behavior of the client system, MonkeyWrench is able to log the exact vulnerabilities that were attacked by a website and extract respective payloads.

Primary contributions of this paper are as follows: An overview on web-based attacks incorporating JavaScript is given in Section 2. We propose a hybrid analysis approach for detecting malicious websites trying to strike a balance between performance and detection rate and describe architecture and implementation of our MonkeyWrench system in Section 3. The system's detection and analysis abilities are demonstrated by evaluation results on the system's performance and results of an examination of several hundred thousand websites as well as an in-depth analysis of a malicious website in Section 4. Related work is discussed in Section 5 and we conclude in Section 6.

2 Web-Based Attacks Using JavaScript

JavaScript enables web designers to create web pages with dynamic content reacting to the user's interactions without reloading the page. Since the script code is downloaded and executed on a client system, attackers are able to go for local vulnerabilities. In addition JavaScript can be used to prepare web-based attacks (e.g. using heap spraying) and to obfuscate the exploit code to prevent static signature-based detection. The following browser vulnerabilities with their "Common Vulnerabilities and Exposures" CVE-ID [2] are subsequently referred to as V1-V6:

V1. Real Networks Real Player CVE-2008-1309: A buffer overflow in Real Player's ActiveX control allows the execution of arbitrary code.
V2. Yahoo! Music Jukebox CVE-2008-0623: A stack-based buffer overflow in the ActiveX control "YMP Datagrid" allows the execution of arbitrary code.
V3. Microsoft Access Snapshot Viewer CVE-2008-2463: A function of the ActiveX plug-in allows the download of an arbitrary file. The exploit may lead to the automatic execution of the downloaded file in a Startup folder.

V4. Microsoft Windows Media Encoder CVE-2008-3008: A stack-based overflow in the ActiveX profile manager component of Windows Media Encoder 9 allows the execution of arbitrary code.
V5. Ourgame GLWorld CVE-2008-0647: Multiple stack-based buffer overflows in the ActiveX control of the chat application Ourgame GLWorld allow remote attackers to execute arbitrary code.
V6. Microsoft Internet Explorer XML Parsing CVE-2008-4844: A vulnerability in Microsoft Internet Explorer allows remote attackers to execute arbitrary code via a crafted XML document.

Many of the commonly exploited vulnerabilities attack ActiveX controls, portable COM-objects used in Microsoft operating systems offering an interface to services complementing the browser's functionality. In this context ActiveX shall be deemed to be a security risk since calls to the controls are executed with the current user's rights. If an installed ActiveX control has a vulnerability like a buffer overflow, the client is exposed to web-based attacks utilizing active content like JavaScript or VBScript. JavaScript sample #1 was found during the experiments presented in Section 4 and aims at exploiting a vulnerability in Real Player's ActiveX-plug-in. The sample shows how attackers utilize JavaScript to customize their attacks to the victim's system at runtime. First of all a Real Player ActiveX-object is initialized (line 1). Afterwards several memory addresses required to successfully exploit the vulnerability are computed using the Real Player's version and the system language of the browser (line 2-17). Then the payload of the attack is assembled using these addresses, a padding and shellcode (line 18). Finally the vulnerable function of the ActiveX control is called with the prepared arguments (line 19). Note that the sample was shortened and essential parts of the script have been omitted to avoid publishing a working exploit in this paper. Section 3.4 takes up the exploitation of this vulnerability again to describe how the analysis modules of MonkeyWrench detect attacks.

JavaScript sample #1 (exploitation of *V1*, omitted details marked with *):

```
 1 real = new ActiveXObject("IERPCtl.IERPCtl.1");
 2 realversion = real.PlayerProperty("PRODUCTVERSION");
 3 Padding = "";
 4 JmpOver = unescape("%75%06%74%04");
 5 if(navigator.userLanguage.toLowerCase() == "en-us")
 6   ret = unescape("%4f%71%a4%60");
 7 else return;
 8 if(realversion.indexOf("6.0.14.") != -1) {
 9   for(i=0;i<10;i++)
10     Padding = Padding + JmpOver;
11   Padding = Padding + ret;
12 }
13 else return;
14 for(i=0;i<32*148;i++)
15   Padding += ******;
16 AdjESP = ******;
17 Shellcode = ******************;
18 PayLoad = Padding + AdjESP + Shellcode;
19 real.import("C:\\WINDOWS\\Media\\ding.wav",PayLoad,"",0,0);
```

3 MonkeyWrench

MonkeyWrench follows a hybrid approach to check web pages by statically analyzing the HTML-objects of a page and running JavaScript in an emulated browser environment for dynamic analysis. Therefore dynamic content can be examined and dynamic redirects can be followed without causing the overhead of a high-interaction web-honeyclient. As opposed to conventional honeyclients MonkeyWrench is able to analyze and identify the techniques used in web-based attacks by checking and logging all calls from JavaScripts to the emulated browser environment.

The MonkeyWrench system consists of two main components that are unified in a console-application written in C++: the client and the analysis engine. As an additional tool WrenchServer was developed as a graphical database user-interface used to visualize the results of an examination offering analysis reports and a source code view of malicious pages and scripts.

3.1 Client

The client represents MonkeyWrench's main program and contains the functionality to locally read or download a web page or a script and examine it using a static analysis and the included analysis engine for dynamic analysis of active content. Figure 2 depicts the workflow of MonkeyWrench's client working off a list of candidates.

First of all the candidate list crafted by the candidate creation component or manually entered by the user is loaded into memory. Afterwards the candidates are either checked one by one or assigned to threads by a taskpool if multi-threading is activated. The candidate's web pages are retrieved via the library libcurl [3] following the Hypertext Transfer Protocol (HTTP) in online mode or read from the filesystem when used in offline mode to analyze network dumps. MonkeyWrench uses the useragent string of Microsoft's Internet Explorer 7 for HTTP requests to avoid simple fingerprinting techniques in online mode. The web pages are subsequently preprocessed and the document tree is being traversed using the libTidy library. JavaScripts are forwarded to the analysis engine where they are executed in an emulated browser environment to enforce behavior based detection of malicious scripts.

Static and dynamic analysis results are managed as events within the client. Events occurring during examination of a candidate are stored for evaluation into a result database using the SQLite-database interface. These events represent interactions of the contents of a page or a script with the emulated browser. During static analysis the web page triggers so-called PageEvents, e.g. induced by an invisible inline frame. JSEvents are the effects of the execution of a script generated in the analysis engine to log calls to methods and properties of browser objects and ActiveX controls in the emulated browser. Every event inside of MonkeyWrench is classified into one of the following five security levels: *erroneous* - event was triggered by an erroneous call or object; *harmless* - event was classified harmless; *questionable* - event could be used to prepare an attack; *suspicious* - event indicates a possible malicious interaction; *dangerous* - event was triggered by interaction used to harm the emulated system. All

events raised in MonkeyWrench by the HTML-parser and the analysis engine are evaluated to determine the security level of the respective website, applying a set of rules. In addition to explicit attacks on the emulated vulnerability modules, MonkeyWrench detects suspicious activities like certain variants of code obfuscation, invisible iFrames and heap spraying techniques.

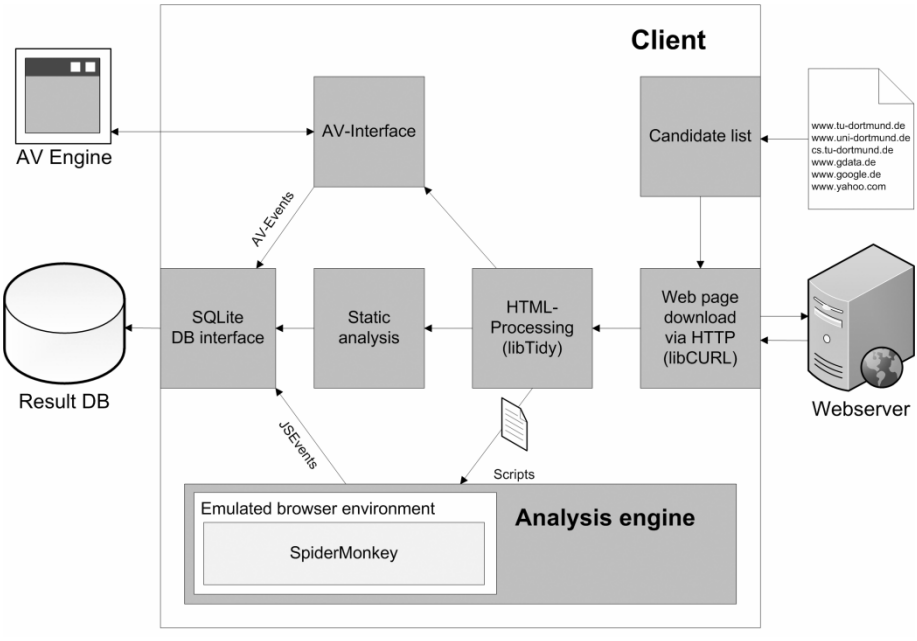

Fig. 2. Workflow of the MonkeyWrench client checking web pages

Static and dynamic analysis results are managed as events within the client. Events occurring during examination of a candidate are stored for evaluation into a result database using the SQLite-database interface. These events represent interactions of the contents of a page or a script with the emulated browser. During static analysis the web page triggers so-called PageEvents, e.g. induced by an invisible inline frame. JSEvents are the effects of the execution of a script generated in the analysis engine to log calls to methods and properties of browser objects and ActiveX controls in the emulated browser. Every event inside of MonkeyWrench is classified into one of the following five security levels: *erroneous* - event was triggered by an erroneous call or object; *harmless* - event was classified harmless; *questionable* - event could be used to prepare an attack; *suspicious* - event indicates a possible malicious interaction; *dangerous* - event was triggered by interaction used to harm the emulated system. All events raised in MonkeyWrench by the HTML-parser and the analysis engine are evaluated to determine the security level of the respective website, applying a set of rules. In addition to explicit attacks on the emulated vulnerability modules, MonkeyWrench detects suspicious activities like certain variants of code obfuscation, invisible iFrames and heap spraying techniques.

3.2 Dynamic Analysis Engine

The task of the analysis engine within MonkeyWrench is to detect active content executing web-based attacks and their preparation, while the scripts are executed in an emulated browser environment. The execution of active content is currently limited to JavaScript using the Mozilla's script engine SpiderMonkey [5]. Thanks to its open-source policy SpiderMonkey could be modified to suit the requirements of analyzing potentially malicious code.

The environment of a web page inside the browser is the Document Object Model (DOM), representing the interface between content and web browser. By reproducing the DOM, MonkeyWrench creates an emulated browser environment to control the calls of a script and enforce the behavior based detection in the client. Every call is checked heuristically for buffer overflows and the occurrence of shellcode to detect unknown and not yet emulated vulnerabilities. Every interaction with the emulated browser triggers an event with a certain security level that is forwarded to Monkey-Wrench's client. A drawback of this approach is that malicious scripts could try to detect MonkeyWrench by calling unimplemented methods to check the return values.

JavaScript sample #2 is part of a malware webkit found on a website during the experiments presented in Section 4. Its task is to deploy different attacks depending on the client's internet browser and installed plug-ins. In the first three lines the script tests the value of *navigator.userAgent* to decide whether or not the browser is Microsoft's Internet Explorer (MS IE) 7. In case that the returned string contains "msie 7" the script loads an iFrame containing an attack against the vulnerability V6. Then the script tries to instantiate the ActiveX-object of V5. Afterwards an iFrame with an attack against this plug-in's weakness is loaded if the browser successfully returns the object. Unlike high-interaction systems MonkeyWrench is able to detect the request for *navigator.useragent* and repeatedly execute the script. Thereby the system is able to react to simple script-side fingerprinting by offering different response values. The prototype of MonkeyWrench currently implements the useragent strings of the following combinations of operating system and internet browser: MS IE 7 on Windows XP, MS IE 6 on Windows XP, MS IE 8 on Windows Vista, Mozilla Firefox 3 on Windows Vista, Mozilla Firefox 2 on Windows XP, Google Chrome on Windows XP, Apple Safari on Windows XP and Opera on Windows XP.

Certainly other JavaScript techniques can be used to fingerprint the client's browser and system and therefore be used to effectively hide malicious activities from MonkeyWrench. One technique to detect MonkeyWrench is to execute certain methods revealing differences in the implementation of the JavaScript engine and the emulated browser environment.

The MonkeyWrench prototype emulates and dependably identifies several ActiveX-vulnerabilities including V1-V6. Furthermore MonkeyWrench can be simply upgraded with additional vulnerability modules. The ActiveX vulnerability modules of MonkeyWrench are triggered by a call to *new ActiveXObject()* or an <object>-tag including the class-id embedded in a web page. MonkeyWrench instantiates the respective module or a general module used to check all method invocations and property requests against arguments containing large strings or shellcode. As an example the class deployed by the vulnerability module "Real Player" (cf. JavaScript sample #1 in Section 2) contains the following methods/properties:

- The Property *Console* vulnerable to buffer overflow induced by setting.
- The Method *PlayerProperty* that returns a string containing the client's Real Player version. The module acts as the vulnerable version "6.0.14.544".
- The method *Import* vulnerable to a buffer overflow induced by a long string passed as the second argument of a call to the method.

During execution of a script the vulnerability modules analyze calls to implemented and unimplemented methods and properties. Alarm events are raised whenever a buffer overflow attempt or an argument containing shellcode is found.

JavaScript sample #2 (part of a malware webkit, shortened):

```
if (navigator.userAgent.toLowerCase()
    .indexOf("msie 7")==-1)
document.write("<iframe src=14.htm><\/iframe>");
try {
  var f;
  var gw=new ActiveXObject("GLIEDown.IEDown.1"); }
catch(f){}
finally {
  if(f!="[object Error]") {
     document.write("<iframe src=newlz.htm><\/iframe>");
  }  }
```

4 Evaluation

This section describes experiments performed to evaluate MonkeyWrench's performance and detection effectiveness in comparison to a high-interaction honeyclient. Furthermore we describe the insights on prevalent web-attacks that we gained during our analysis of web pages and we present an in-depth analysis of one of the malicious web pages. The experiments described in this section were performed from May 25th to June 6th 2009 on a 2.4 GHz Intel Core-2-Quad test system with 8 GB of RAM running Debian Linux "Lenny" connected to the internet with 2 MBit/s DSL. For comparative evaluation the high-interaction honeyclient CaptureHPC [1] version 2.51 was installed on the test system running a Windows XP SP2 client with Internet Explorer 6 using the default configuration.

4.1 Multi-threading Performance

As mentioned in Section 3, MonkeyWrench is able to check multiple pages concurrently using multithreading. The diagram in Fig. 3 shows how multithreading affects the average length of an experimental web page check on our test system during an examination of the top 1000 domains of the Alexa top list worldwide [11].

Since the DSL connection presumably acts as the restrictive factor in this experiment a reevaluation utilizing a better internet connection is considered. During the examination one site that tried to instantiate a spyware plug-in was found to be malicious by MonkeyWrench.

CaptureHPC finished the list of Alexa's top 1000 domains in 212 minutes, which equals an average check length of 12.72 seconds per page. No malicious pages were found and the virtual machine image was automatically set back 26 times during the process. CaptureHPC was using the default configuration of 20 concurrent Internet Explorer processes with a visitation time of 10 seconds after each page was loaded.

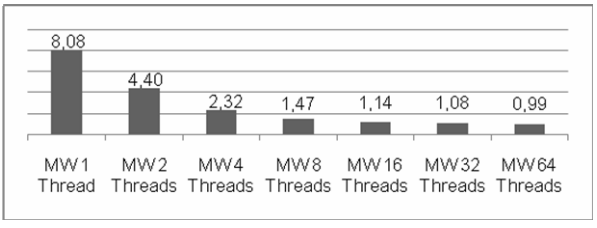

Fig. 3. Average length of a web page check in seconds during examination of Alexa Top 1000 domains world-wide using the MonkeyWrench prototype (average value of 3 runs)

4.2 Web Attacks

During the examination a total of 567.157 web pages were checked. The candidate list was created by querying Google's search engine with promising search terms and URLs that were reported by users. MonkeyWrench found 31.817 malicious web pages equaling a rate of about 5.6% and 19.317 inline frames invisible to the user pointing to malware distribution pages. During the examination the system saved approximately 3 GB of HTML, JavaScript (obfuscated and deobfuscated) and binaries including 2.114 unique (differing MD5 values) malicious executable samples.

All files generated by MonkeyWrench were scanned utilizing the G Data Linux antivirus engine. The scanner marked a total of 43.175 files as malicious. The bulk of the antivirus detections were triggered by files that were deobfuscated by the MonkeyWrench system. Therefore a HTTP scanner as utilized by many common antivirus solutions would not have detected these attacks since the attacks are dynamically decrypted in the browser. A noticeable amount of detections were triggered by signatures not targeting web-based exploit code but inline frames pointing to known (blacklisted) malware distribution domains.

Several large-scale attacks were identified using the result database of MonkeyWrench. Thereby several thousand infected pages were linked to the malware distribution servers used. The largest attack found by MonkeyWrench was the infection of 887 web pages using SQL injection. The actual attack was carried out by a malware server having a DNS name with the top level domain of China using 33 different domains. A total of 598 malicious pages were trying to lure the user into installing a fake media codec being a trojan horse. An attack against a recent Adobe Acrobat Reader vulnerability (CVE-2009-0927 [2]) infected 391 domains hosting adult content. Another infection, called Gumblar/Martuz used websites that were taken over by attackers through stolen FTP credentials and vulnerable web applications. MonkeyWrench detected 181 infections of Gumblar/Martuz.

The majority of malicious pages tried to exploit more than one vulnerability. Figure 4 depicts the distribution of the attacks detected by MonkeyWrench's ActiveX

vulnerability modules. The numerous exploitation of the Microsoft Access Snapshot Viewer vulnerability may be explained by its ease of use: The attacker just has to use a specific regular functionality of the ActiveX control to download an arbitrary remote file to the local filesystem and to overwrite system files.

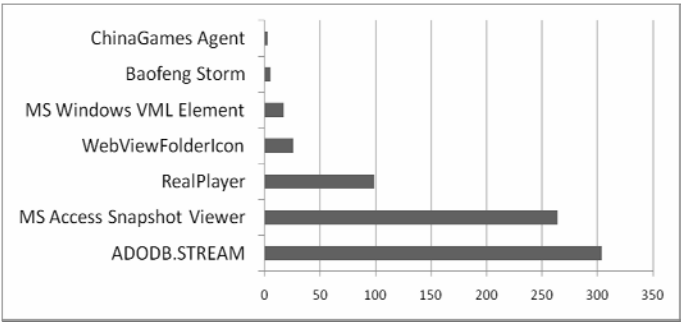

Fig. 4. Detected exploitation of vulnerabilities

A fraction of 900 web pages were reexamined to compare the results and in particular the detection rates of MonkeyWrench and CaptureHPC. Of the 900 pages checked MonkeyWrench found two pages to be malicious while CaptureHPC marked all pages as benign. The results of MonkeyWrench revealed that the malicious websites both attacked vulnerability V1. Therefore the virtual machine image used in CaptureHPC was equipped with a vulnerable version of “Real Player”. In a subsequent run CaptureHPC detected the malicious pages previously found by MonkeyWrench but had no further findings.

4.3 In-Depth Analysis of Web-Based Attacks

In the following we analyze the techniques of one particular malicious page found during our examination and exemplify MonkeyWrench's ability to collect samples of partly unknown malware and JavaScript exploit code. The page offers an overview of the server status in the online role-playing game “World of Warcraft” to users. The market leader in online role-playing games “World of Warcraft” had over 11 million subscribers at the end of 2008 with a sales volume of about one billion dollars. It is uncertain whether the website's server was compromised by an attacker or deliberately prepared by the operator to lure users into visiting a malicious page. Embedded inline frames pointing to a malicious distribution site were repeatedly found during examination of the page. The inline frame redirected to content hosted by “pagead2.googlesyndicastion.com”. This domain resembles the URL of Google's advertisement application AdSense “googlesyndication.com”. Presumably the attacker tries to obfuscate the injected inline frame so that it is ignored by web admins during the manual inspection of the page's source code. The malicious content hosted by the distribution site tries to exploit several browser- and plug-in-vulnerabilities namely V1, V3 and V6. A JavaScript first checks the name and version of the user's browser and accordingly reloads adjusted variants of the Internet Explorer exploit.

Afterwards the system is tested for installed plug-ins and the corresponding exploits are loaded into the browser. During the examination of the page, MonkeyWrench collected a malware sample that ought to be installed by the distribution site. The G Data anti-virus-scanner used in the test system did not identify the sample as malware. Therefore the sample was uploaded to the web service Virustotal.com and scanned using anti-virus-scanners of 39 vendors. Only three engines identified the sample as malware: Prevx1: Malicious Software; Rising: Trojan.PSW.Win32.WoWar.ayg; SecureWeb-Gateway: Win32.LooksLike.Rootkit.gen. Prevx1 and SecureWeb-Gateway seemed to identify the sample using generic signatures whereas Rising could identify the sample more precisely. A description on the behavior of "Trojan-.PSW.Win32.WoWar.ayg" could not be found, thus it was executed in the sandbox application CWSandbox [12] to find out how it would affect a user's system. After installation the malware would copy itself to the system folder and register as a system service to automatically run at startup. In a manual comparison of the malware's behavior and characteristics of known malware samples, the sample could be assigned to the malware family "Trojan-GameThief.Win32.OnLineGames". The malware is a polymorphic trojan horse thus having a different MD5-checksum than other samples. Trojan horses of this family are used to record the user's keyboard input and monitor running processes. Once the user of an infected system starts "World of Warcraft" or other online games, the malware logs the account information and sends it to a remote server via HTTP. Therefore the malware used by the attacker is directly connected to its distribution since almost only customers interested in "World of Warcraft" are visiting the compromised website.

5 Related Work

Provos and Holz [9] provide a broad introduction to the subject of honeypots. There are several ongoing projects engaging in the development of web-honeyclients. Kathy Wang released an open-source honeyclient in 2005 whose development is nowadays coordinated by the MITRE Honeyclient Project [4]. It consisted of two Perl-scripts managing an instance of Microsoft's Internet Explorer and monitoring the system while web pages are checked. Provos et al. [8] conducted a comprehensive study of malware on the web by using honeyclients to check the pages of Google's web repository. Microsoft uses Strider HoneyMonkeys [10], high-interaction web-honeyclients based on virtualized Windows operating systems, to detect the abuse of previously unknown vulnerabilities. CaptureHPC [1,7] is a high-interaction web-honeyclient released as open-source software by Seifert et al., those who earlier published their work on the HoneyC system [6] following a low-interaction approach. While effective, existing honeyclients are limited either in run-time performance due to their high-interaction approach or in detection performance due to their inability to dynamically analyze JavaScripts. In contrast to that, our honeyclient MonkeyWrench dynamically analyzes JavaScript in an emulated browser environment while avoiding run-time performance overheads by realizing a low-interaction approach.

6 Conclusion and Future Work

The evaluations in this paper demonstrate the abilities of MonkeyWrench as an efficient tool to examine the techniques of attackers using malicious web pages and to collect previously unknown malware and JavaScript exploit samples. Besides detecting web-based attacks based on events triggered by the vulnerability modules, MonkeyWrench uses advanced techniques like shellcode and heap spraying detection to spot previously unknown attacks. Compared to the high-interaction honeyclient CaptureHPC, MonkeyWrench showed a considerable advantage in performance during our experiments. MonkeyWrench's emulated browser environment allows a researcher to identify and display exactly which vulnerabilities are attacked by a malicious website and how the attack was prepared. Certainly the approach has some drawbacks naturally caused by the emulation of a system like attackers who could avoid being detected performing either server- or client-side fingerprinting.

The in-depth examination of websites proves that the attackers adjust their attacks to capitalize on booming trends within the internet community. Users are lured into visiting web pages that contain promising content though attacking the browser to perform drive-by downloads of customized Trojan horses which typically steal the user's account information and password that is afterwards resold by the attacker.

The web-honeyclient MonkeyWrench represents a prototype that features some aspects that need further development to achieve the approach's full potential. Thus the execution of active content besides JavaScript like VBScript or Adobe Flash including Actionscript is considered.

References

1. The Honeynet Project CaptureHPC, `https://projects.honeynet.org/capture-hpc/`
2. Common Vulnerabilities and Exposures (CVE), `http://cve.mitre.org/`
3. libcurl, `http://curl.haxx.se/libcurl/`
4. MITRE Honeyclient Project, `http://www.honeyclient.org/trac`
5. Mozilla SpiderMonkey (JavaScript-C) Engine, `http://www.mozilla.org/js/spidermonkey/`
6. Seifert, C., Welch, I., Komisarczuk, P.: HoneyC - The Low-Interaction Client Honeypot. In: Proc. of the 2007 NZCSRCS. Waikato University, Hamilton (April 2007)
7. Seifert, C., Steenson, R., Holz, T., Yuan, B., Davis, M.A.: Know Your Enemy: Malicious Web Servers (August 2007), `http://www.honeynet.org/papers/mws/`
8. Provos, N., Mavrommatis, P., Rajab, M., Monrose, F.: All Your iFrames Point to Us. In: Proc. of the 17th USENIX Security Symposium (July 2008)
9. Provos, N., Holz, T.: Virtual Honeypots: From Botnet Tracking to Intrusion Detection. Addison-Wesley Professional, Reading (2007)
10. Wang, Y., Beck, D., Jiang, X., Roussev, R., Verbowski, C., Chen, S., King, S.: Automated Web Patrol with Strider HoneyMonkeys. In: Proc. of NDSS 2006 (2006)
11. Alexa the Web Information Company - Top Sites, `http://www.alexa.com/topsites`
12. CWSandbox – Automated Malware Analysis, `http://www.cwsandbox.org/`