

# MARIONETTE: Client Honeypot for Investigating and Understanding Web-based Malware Infection on Implicated Websites

Mitsuaki Akiyama, Yuhei Kawakoya, Makoto Iwamura,  
Kazufumi Aoki, Mitsutaka Itoh

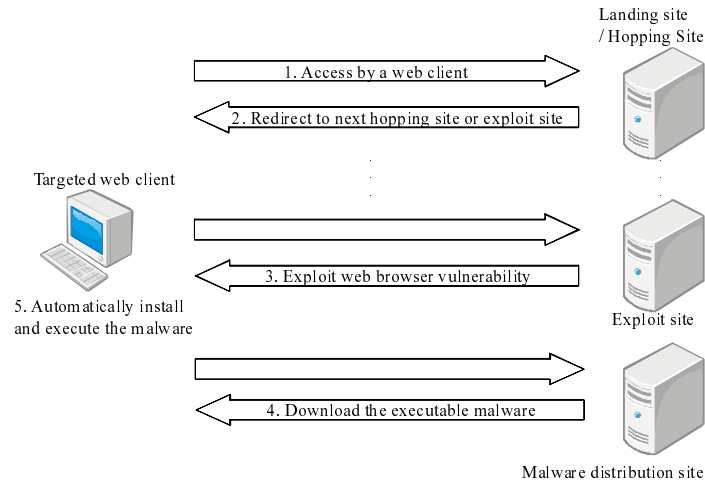
NTT Information Sharing Platform Laboratories, NTT Corporation, Tokyo, Japan,  
{akiyama.mitsuaki, kawakoya.yuhei, iwamura.makoto,  
aoki.kazufumi, itoh.mitsutaka}@lab.ntt.co.jp

**Abstract.** Nowadays, the number of web-browser targeted attacks that lead users to adversaries' web sites and exploit web-browser vulnerabilities is increasing, and a clarification of their methods is urgently needed. In this paper, we introduce our design and implementation of a client honeypot with the capacity for gathering information, and we describe our investigation, conducted over a prolonged period, of web-based malware and its related malicious web sites. The results reveal not only detected malicious web sites and analyzed collected malware but also the anomalous network structure of malicious web sites composed by various redirection methods.

## 1 Introduction

Nowadays, malicious programs called malware are spreading widely over the Internet and causing a variety of security incidents for infected hosts. Infected hosts become part of the malicious infrastructure that causes DDoS attacks, mass-mailing, infection activities, and other malicious activities. Therefore, now that malware is a main threat on the Internet, immediate countermeasures are needed.

One factor of malware spreading is the diversification of its infection vectors. Server-side attacks that target OS or server application vulnerabilities are commonly used by adversaries. In contrast, client-side attacks are triggered by a target action. For example, an adversary lures a target client to his/her web site and exploits a vulnerable client's application (e.g., web browser) to download malware to the targets' PCs. An adversary often exploits web browsers in combination with defacing the web contents of compromised web servers [8] [11]. These attacks are often called *drive-by-download* because they automatically and invisibly install malware once a web client visits the web sites. Moreover, these attacks cross network boundaries and pass through network boundary protection, such as FW or IDS/IPS, and undermine those inside target networks. The increasing number of these incidents necessitates countermeasures at both the client side and network side against malware. To protect web users at the client side, we need to comprehend the nature of web-based malware and the latest attack tendencies, such as targeted vulnerabilities, and apply countermeasures, such as creating an anti-virus tool's pattern file and prompting users to apply a certain security patch and so on. To protect web users at the network side, we need to detect malicious sites in web space and prevent web clients from accessing to those sites and being exploits. According to the field investigations conducted in previous studies on web-based malware [4] [6] [7] [17], malicious web sites connect with each other by redirection mechanisms and invisibly attack target clients and install malware to their PCs. However, at present the vulnerabilities that tend to be targeted and the structure of malicious sites chained by redirection are not clear. Thus, we need investigation tools to collect the above information.



**Fig. 1.** Sequence of web browser exploitation

Honeypots are used for investigating and collecting information on the exploitation methodology of adversaries and malware executables. Conventional honeypots are ready and waiting for server-side attacks and thus detect them. However, client-side attacks are triggered by target actions, therefore these attacks are outside the scope of these honeypots. Thus, an active honeypot compatible with client-side attacks is needed. This type of honeypot is called a client honeypot. Various architectures and implementations of client honeypots are proposed in Refs. [3] [15]. However, the capability for detecting attacks and collecting information by each architecture is not sufficient because of the inaccuracy of collected information, unstable running (such as hijacking by adversaries), and low performance of running. For comprehending exploitation techniques, the nature of new variants of malware, and the redirect chain of malicious sites, we designed and implemented a client honeypot that detects web browser exploitation in accurate detail. We investigated malicious web space for a long period using our system and analyzed the collected information about exploitation and malware executables.

The remainder of this paper is organized as follows. Section 2 introduces the principal methodologies of web browser exploitation. In Sect. 3, we describe the architecture of existing client honeypots. In accordance with their advantages and disadvantages, we designed and implemented a new client honeypot. In Sect. 4, we investigate malicious web servers and web-based malware using our client honeypot in a field investigation. Our results reveal not only the natures of detected malicious web sites and malware but also the anomalous network structure of malicious web sites.

## 2 Methodologies of exploitation

In this section, we introduce some methodologies of web browser exploitation to understand the drive-by-download attacks shown in Fig. 1. Descriptions of the main methodologies follow.

**Redirection to exploit sites by hopping sites** In most situations, malicious sites are divided their responsibilities [6] by an adversary. An exploit site does the actual exploitation of a target

web browser and forces it to download a malware executable from a malware distribution site. A hopping site redirects a target to the next hopping site or an exploit site. A hopping site that is accessed first is called a landing site. These sites are illustrated in Fig. 1. In this paper, we call this redirection network a malware distribution network.

First, adversaries lure web clients to exploit web sites using compromised web sites that are injected with a malicious iframe tag or JavaScript tag. Once web clients access compromised web sites, they are redirected automatically to the next hopping site or an exploit site. A hopping site that is injected with a redirect instruction code, in some cases prepared by adversaries themselves, is chained to the next hopping site or the exploit site. In the case of accessing only a landing site, the web client (web browser) is forced to access the exploit site eventually due to the malicious redirect instructions. Adversaries use this method because it can widely capture a lot of web clients, evade attack detection, and make it more difficult to track the adversaries. Representative redirection methods are protocol redirect (HTTP 30x), tag redirect (frame tag, iframe tag, META tag set *refresh* attribute, etc.), and script redirect (JavaScript-*location.x* method etc.). In particular, invisible redirect contents (e.g., a iframe that has small pixel height/width and an invisible attribute) are often used for avoiding notice by users.

**Sequential attacks targeting multi-vulnerabilities** A number of web browser exploiting techniques sequentially attack multiple vulnerabilities to increase the possibility of a successful exploitation. Web browsers and its plug-ins have a variety of vulnerabilities (e.g., buffer-overflow, heap-overflow and so on). A well-known toolkit called MPack [12], which can easily create an exploit site, typically performs sequential attacks. MPack contains components that can attack critical vulnerabilities that are mainly involved in Internet Explorer and its plug-in applications. Many exploit sites created using MPack have been reported [11], and we also observed a number of malicious web sites using MPack, described in Sect. 4

**Exploit-code obfuscation** Malicious sites often have obfuscated malicious JavaScript codes, which are exploit codes or redirect tags to the exploit site, in order to elude IDS signatures and make analysis more difficult. A loaded, obfuscated JavaScript is rendered by the scripting engine of a web browser and a de-obfuscating calculation (e.g., *unescape()* function, hex escaping, string replacing, XOR, and so on). Obfuscated JavaScript includes not only an exploit code but also a redirect tag (e.g., iframe tag). However, obfuscation for protecting JavaScript code is also used by many general sites. For this reason, detecting malicious sites by detecting only obfuscated JavaScript includes false-positives. Therefore, we need to use real web browsers and run web contents on it to detect malicious web sites accurately.

### 3 Client honeypot

In this section, we enumerate the requirements for a client honeypot for an investigation into malware and malicious web sites, and introduce the characteristics of existing client honeypots. In accordance with these requirements, we designed and implemented a new client honeypot.

#### 3.1 Requirements

The goal of our field investigation was to understand the methodologies and purpose of adversaries and to eventually provide countermeasures based on collected information. Therefore, a system is needed to collect the requisite information: malicious URLs, type of vulnerabilities

exploited, malware executables, nature of malware, and related nodes of malware distribution networks.

To satisfy the above requirements, we focused on the points to design and implement as follows.

- **Detection accuracy:**  
For reducing false-positives and false-negatives in detection, a client honeypot should detect various attacks accurately.
- **Detection variety:**  
A client honeypot should detect wide range of attacks, not only attacks to well-known vulnerabilities but also zero-day attacks.
- **Collection variety:**  
It is needed for countermeasures that collecting more information for not only binary decision whether or not a target site is malicious but also targeted vulnerabilities, redirect URLs, nature of malware and so on.
- **Performance Efficiency:**  
To patrol the very large web space, it is necessary to efficiently test a large amount of web pages.
- **Safety and stability:**  
For preventing hijacking and maintaining continuous running, a client honeypot should sustainably detect attacks and collect malicious executables generated by drive-by-download attacks without being compromised and destroying the honeypot environment.

### 3.2 Related works

A client honeypot that focuses on the web is a system that automatically patrols the web and detects malicious web sites. The system is composed of two main components: a web patrolling function and an attack detecting function.

Client honeypots can be categorized into two types: high interaction and low interaction. The former uses a real web browser, and the latter uses a browser emulator to patrol the web. Because a real browser is used, rich information can be collected by a high interaction system (e.g., behavior after exploiting a system, malware executables created after exploitation, etc.). However, a real browser risks infection because of running exploit codes on it. In contrast, a low interaction system has no risk of infection, whereas it collects less information than the high interaction system because exploit codes cannot run on browser emulators. Moreover, its detection method, which uses signatures or anti-virus pattern files to check web contents, often produces false-positives (deciding a legitimate web page is a malicious web page) and false-negatives (deciding a malicious web page is a legitimate web page).

Capture-HPC [5] and HoneyClient [3] are well-known high interaction client honeypots that can detect the malicious activities of malware after PC infection regardless of a vulnerability category. Thus, these systems cannot detect attacks in the case of a failed exploit attempt. Moreover, these systems become infected and perform malicious communications with other hosts after exploitation such as secondary infection, though these are the typical aspects of a high interaction system. In addition, HoneyClient's detection method, based on integrity checking of the file system, has a high computational cost and a latency of several minutes per web page. These systems have performance issues during the inspection because they can only run one process on one OS. Additionally, each of these client honeypots cannot extract the redirect chain of malware distribution networks and can only extract the first accessed URL.

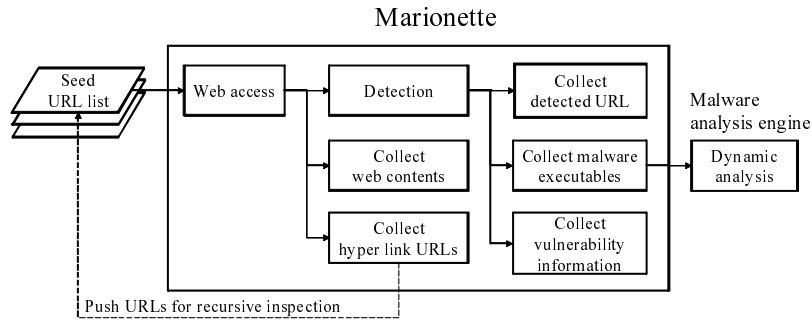


Fig. 2. Workflow of our system

### 3.3 Proposed architecture

Our proposed client honeypot, called MARIONETTE, has a high interaction architecture and provides a capability of collecting detailed information. A basic workflow of MARIONETTE is shown in Fig. 2. First, it accesses web pages based on a seed URL list. Next, in the case of an exploitation occurring, the system collects accessed URLs, the category of vulnerability, and the malware executable. The URL list is previously compiled from search engine results, URL extraction from spam mail, and so on. Collected malware are analyzed by our malware analysis engine in collaboration with MARIONETTE. MARIONETTE provides three main functions: 1) accurate exploit detection and malware executable collection, 2) continuous running and prevention from hijacking, and 3) redirect extraction of malware distribution networks. We describe the methodologies of exploit detection and of extraction of chains of malware distribution networks.

### 3.4 Detection methodologies

The proposed system has stepwise detection mechanisms: JavaScript-anomaly detection, vulnerability-specific detection, and local resource monitoring. These mechanisms work at various stages of attacks. JavaScript-anomaly detector focuses on the discriminative malicious behavior (e.g., heap spraying<sup>1</sup>) of JavaScript before it exploits a certain vulnerability. Vulnerability-specific detector focuses on the moment a vulnerability is exploited. Local resource monitoring focuses on the behavior of *shellcode*<sup>2</sup> on a web browser or browser helper object (BHO) after exploiting a certain vulnerability. These detection mechanisms are next described in more detail.

**Script-anomaly detector** Client-site scripting such as JavaScript and VBScript on a web browser has a lot of flexibility and provides rich and dynamic web contents. However, runtime deobfuscation of malicious code, shellcode injection to the memory of a web browser, and sequential attacks can often be carried out through JavaScript or VBScript. This detection method monitors scripting engines on a browser and detects anomalous behavior under the condition of attacks.

One instance under the condition of attack is heap spraying [18]. Heap spraying injects a vast amount of malicious instruction code blocks to the heap memory of a target web browser prior to

<sup>1</sup> injecting a vast amount of malicious instruction code (i.e., sliding code and shellcode) to the heap memory of the target web browser to improve the possibility of hijacking the target system

<sup>2</sup> malicious short instructions for execution after hijacking of target system

**Table 1.** Covered vulnerabilities

Vulnerability ID	Vulnerable point
MS06-001	Windows Meta File (WMF)
MS06-014	Microsoft Data Access Component (MDAC)
MS06-055	Vector Markup Language (VML)
MS06-057	Web View Folder Icon
MS07-004	VML
MS07-009	MDAC
MS07-017	Animation cursor
CVE-2006-5198	WinZip plug-in
CVE-2007-0015	QuickTime plug-in

an exploit attempt in order to ensure hijacking of the target system. This technique has versatility, so it is combined with various types of exploitation, because it is possible to exploit by only enforcing instruction pointer to point somewhere in heap memory of scripting engines. We focused on the behavior of heap spraying, and we implemented the detecting function based on a heap memory anomaly, which is the injection of a large amount of strings that include large sliding code and small shellcode blocks. Exploit codes packed in MPack that use heap spraying allocate about 80 to 200 MB or over of strings. Allocated string length is different in each vulnerability though, sprayed large heap blocks should be allocated from certain address on heap memory to the location pointed by overwritten return address in order to run a shellcode. In contrast, normal web pages allocate from a few bytes to several megabytes at most. Thus, we decide the heuristic threshold of total allocated heap block size for heap spraying detection. Additionally, this detector can extract shellcode from sprayed heap blocks by hooking heap allocation function regardless of whether the attack is success or not.

**Vulnerability-specific detector** This detection module monitors certain vulnerable points of web browsers and BHOs. When the instruction pointer of a web browser passes through those points, the module detects and output the logs. We implemented this monitoring on vulnerable versions of web browser components and BHOs. For example, the condition for buffer-overflow detection is whether or not the provided string length is over the buffer size. This detection module is composed small piece of code and is mapped to the target process memory space at a vulnerable function and intercepts vulnerable function calls. This detection module observes the arguments data and the results of calculations, and if a malicious arguments data is passed or an unexpected result is returned, these function calls are recognized as an attack. Covered vulnerabilities are shown in Table 1. Almost all of these can be attacked by components of the MPack toolkit. Our system achieve to accurately detect malicious web sites constructed using the toolkit or a part of it. In addition, our system is able to detect attacks even if they fail to exploit because the exploit code passes through the vulnerable point. In this way, our system bring to recognize the category of exploited vulnerability and detect unstable exploitation. Our implementation referenced released vulnerability information and a third-party patch [19] that discloses vulnerable points and security patches by assembly code level.

**Process Sandbox** Local resources, such as the file system, process space, and registry, are monitored and controlled by process sandbox. It monitors and restricts creating a malware executable, creating process of malware and accessing to critical registries by hooking certain API calls

(e.g., *NTCreateFile()*, *CreateProcess()*, *SetRegKey()*, etc.) to prevent infection and keep the system continuously running. Malicious created processes and files that are not created by the web browser in normal situation are restricted. In particular, in the case of a malicious process trying to create files in a certain directory, file creation is restricted and the file is copied to a sandbox directory, while process sandbox return the success values of the API calls to API caller in order to hide the honeypot aspect from adversaries' side. We examined the behavior of web browsers and BHOs in advance to create a white list of legitimate behavior in order to reduce false-positives.

Process sandbox separates a target process from local resources, enables each process to perform independently. Capture-HPC has a similar detection module, but could not perform by multi-process and by only one browser process on each OS because this module monitors whole of the system at the kernel mode level, while our system monitors each process at the user mode level. Thus, each patrolling process (e.g., web browser and BHO) is able to crawl web space simultaneously. The number of crawler processes depends on machine's capacity, Marionette is able to run 10 crawler processes or over simultaneously on our experimental machine mentioned in Sect. 3.6. Consequently, the inspection performance of our proposed system is ten times or more faster than that of existing high interaction client honeypots. In this way, we improved performance, safety and stability of high interaction system.

### 3.5 Extract redirection of malware distribution networks

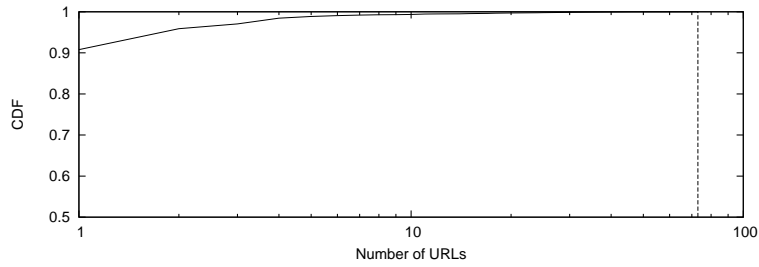
Malicious sites are often structured in multiple stages using redirection, but existing client honeypots cannot extract the site to which the redirection leads (only extract the first accessed sites) as previously mentioned in Sects. 2 and 3.2. HoneyMonkeys [2] also tried to extract these redirect chains, however it provided small set of data and only tracked source-destination URLs. To track large scale exploit sites in details, we implemented a function to extract the source-destination URLs and the category of redirection in concurrence with patrolling. For example, iframe redirection from *www.xxx.com* to *www.yyy.com* is represented by (*www.xxx.com*, *www.yyy.com*, *iframe*). We can extract 3-tuples even supposing that multi-redirection occurs.

We combined two methods, which are HTTP access trapping and Document Object Model (DOM) parsing. The former traps HTTP access and checks the destination URL of an HTTP request and referrer, which is a source URL. The latter searches the redirect tag from a DOM tree on a web browser after de-obfuscating JavaScript and extracts the destination URL (i.e., targeted URL by *src* attribute) and source URL (i.e., frame URL). In the case of an unknown redirect, the attribute is set to *unknown*. The reason we combined two methods is that often one method is not sufficient to extract 3-tuples; the former is not effective in the case of a void referrer by script redirect and protocol redirect, while the latter is not effective in the case of a event-driven or delayed rewriting DOM object.

In this way, this function is compatible with the redirections described in Sect. 2. However, this function cannot extract malware distribution sites because accessing such sites is not redirection. Thus, to track those sites, we should combine the functions with a traffic log.

### 3.6 Implementation

The platform of our proposed client honeypot is Internet Explorer 6.0 on Windows XP SP2. Because Internet Explorer 6.0 includes exploitable vulnerabilities and almost all these vulnerabilities can be attacked by MPack, it is suitable for a basis of honeypot system. For this reason, we adopted the platform and implemented on it the proposed client honeypot. Additionally, vulnerable versions of QuickTime 6.5.2 and WinZip 10.0 plug-in were installed.



**Fig. 3.** Cumulative fraction of number of URLs on each FQDN

## 4 Investigation

The proposed client honeypot system MARIONETTE was used for an investigation of web-based malware. In this section, the analytical results of our investigation are described from the viewpoint of the natures of the detected web sites and collected malware executables.

### 4.1 Method of investigation

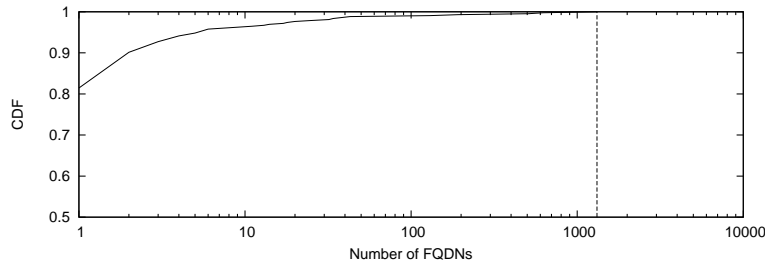
Our investigation by the field trial targeted certain URLs that were preliminarily determined as malicious URLs. We applied malicious seed URL lists<sup>3</sup> including about 300,000 URLs. Our investigation was conducted periodically from February to August, 2008. We recorded detected URLs and IP addresses, redirect URLs from first accessed URLs, malware executables and so on for in-depth analysis.

### 4.2 Nature of malicious sites

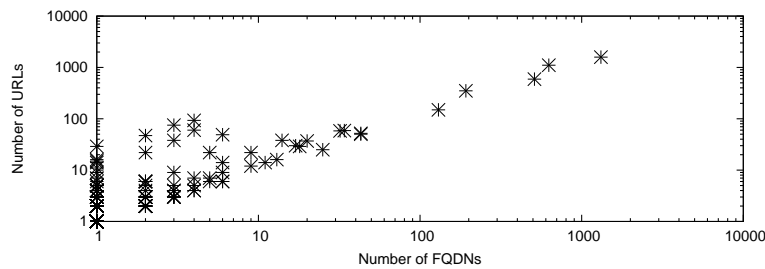
We checked the relations between the URL, FQDN, and IP address of detected malicious web sites. We did not distinguish whether detected sites were landing sites, hopping sites, or exploit sites. There were 5,770 unique URLs<sup>4</sup>, 2,130 unique FQDNs, and 644 unique IP addresses involved with the detected sites. Certain FQDNs had multiple malicious URLs, such as the file hosting site shown in Fig. 3. Likewise, certain IP addresses had multiple FQDNs, such as the hosting server shown in Fig. 4. In particular, large hosting servers hosting malicious sites including hundreds to 1,300 FQDNs were observed. The number of FQDNs and URLs on each IP address are shown in Fig. 5. This figure represents the types of hosting: 1) one IP address with one URL on one FQDN, 2) one IP address with multiple URLs on one FQDN, 3) one IP address with one URL on each of multiple FQDNs, and 4) one IP address with multiple URLs on multiple FQDNs. These types have advantages and disadvantages with regard to manageability, operability, and difficulty of countermeasures for adversaries. Types 3) and 4) require the maintenance cost of a domain name, but on the other hand they can easily evade listing on a URL blacklist. In the case of running many web servers of type 1), an adversary must pay additional server maintenance costs. We assume adversaries choose applicable types of hosting for themselves in consideration of the associated cost and trouble.

<sup>3</sup> Malicious seed URL list provided by Microsoft Corporation Japan in association with NTT Communications Corporation, and URL black list published on the web

<sup>4</sup> We removed URL parameter from these URLs



**Fig. 4.** Cumulative fraction of number of FQDNs on each IP address



**Fig. 5.** Distribution of number of URLs and FQDNs on each IP address

### 4.3 Targeted vulnerability tendency

The proposed system is able to detect exploits at each vulnerability because it monitors the vulnerable point and detects if that point is passed, as mentioned in Sect. 3.4. Statistics of the targeted vulnerabilities of each detected attack are shown in Table 2. The summation of the percentages is over 100% because of sequential exploiting, as previously mentioned, and multiple iframe tags embedded in contents redirecting to multiple exploit sites.

In particular, almost all the exploit sites targeted MS06-014 that is a vulnerability of Microsoft Data Access Component (MDAC). The reason that the distribution of targeted vulnerabilities is biased seems to be that sequential attacks first target MS06-014 because of the high success rate of exploitation, high controllability after hijacking and no need to heap spraying. In this regard, if we use MDAC-patched MS06-014, other vulnerabilities may be targeted second.

### 4.4 Nature of web-based malware

We collected about thirty thousand malware executables in this field investigation. For reducing false-positives, where legitimate files are identified as malicious by mistake, we differentiated files according to whether or not they were created after a certain vulnerability was exploited. Then we extract suspicious format files (e.g., *.exe*, *.sys*, *.scr*, *.vbs*, *.bat* and so on). When malware executables are classified by SHA1-hash values, there are only 695 unique binaries. The reason that the number of unique binaries is only 1/40 of the number of collected malware binaries is mentioned in Sect. 4.5.

We conducted anti-virus (ClamAV [1]) scanning and behavior analysis using our analysis tool, which executes malware executables on a virtual environment and monitors their communication. We compared the executables collected from February to August 2008 in September 2008

**Table 2.** Statistics of targeted vulnerabilities

Vulnerability ID	Percentage
MS06-001	3.5
MS06-014	96.2
MS06-055	1.2
MS06-057	0.7
MS07-004	0.1
MS07-009	0.0
MS07-017	14.2
CVE-2006-5198	0.2
CVE-2007-0015	0.1

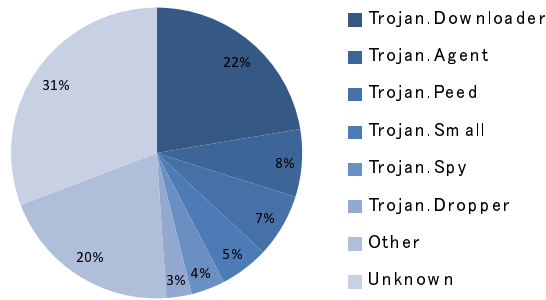
to the latest anti-virus pattern file. The results are shown in Fig. 6. Of the executables, 69% were identified as a variant of malware. In other words, 31% of the executables were false-negatively identified as normal. What is more, in the case of anti-virus scanning straight after collection, the identification rate was less than 40%. The rate depends on the period from discovering a malware and creating an anti-virus pattern file to publicly releasing the file. Many identified malware are categorized as *Trojan-Downloader* or variants of it. Trojan-Downloader has a simple download function, and it downloads main components (e.g., mass-mailing module, DDoS module, information stealer, etc.) to the target's PC. Thus, analyzing only a downloader itself is insufficient for revealing the primary aim of an adversary.

Next, the malware was analyzed by our dynamic behavior analysis engine, which executes malware and monitors its communications. This system has two environments: a closed environment and half-open environment. The former includes fake servers (i.e., DNS, IRC, HTTP) emulating the real Internet in a virtual environment, and the latter connects to the real Internet according to need (e.g., to download a additional component). In this analysis, each malware was analyzed for 3 minutes in a closed environment. The result of dynamic behavior analysis of collected malware executables is shown in Table 3. Of the malware, 57% communicate with other hosts, and 96% of those 57% communicate by HTTP. This result shows a distinction of malware based on infection vectors. Generally, it said that typical malware such as a bot [10] [13] [14] uses Internet Relay Chat (IRC) as its main communication protocol, while web-based malware mainly uses HTTP. The reason web-based malware uses HTTP is that it is commonly used by Internet users and is easy to camouflage as a normal communication to evade detection. Communications to other hosts involve downloading main components by downloader and sending compromised hosts' information to adversaries' servers.

We extracted 349 communication hosts (239 FQDNs and 110 IP addresses) after infection. Some different malware executables communicate same destination hosts and same communication pattern (e.g., protocol, payload strings). One of the occasions is that these malware executables derived from multiple exploit sites made by same adversary. Therefore, we assume that specific communication patterns enable rough classification of malware executables for simplifying malware analysis based on the similarity of communication pattern in advance of high cost and in-depth analysis such as reverse code engineering, long-term dynamic analysis and so on.

#### 4.5 Tracking malware distribution networks

To track malware distribution networks, our system extracts the relations between each URL on those networks. We extracted and analyzed the redirect chain intended for detected URLs.



**Fig. 6.** Variants of malware

**Table 3.** Result of dynamic behavior analysis

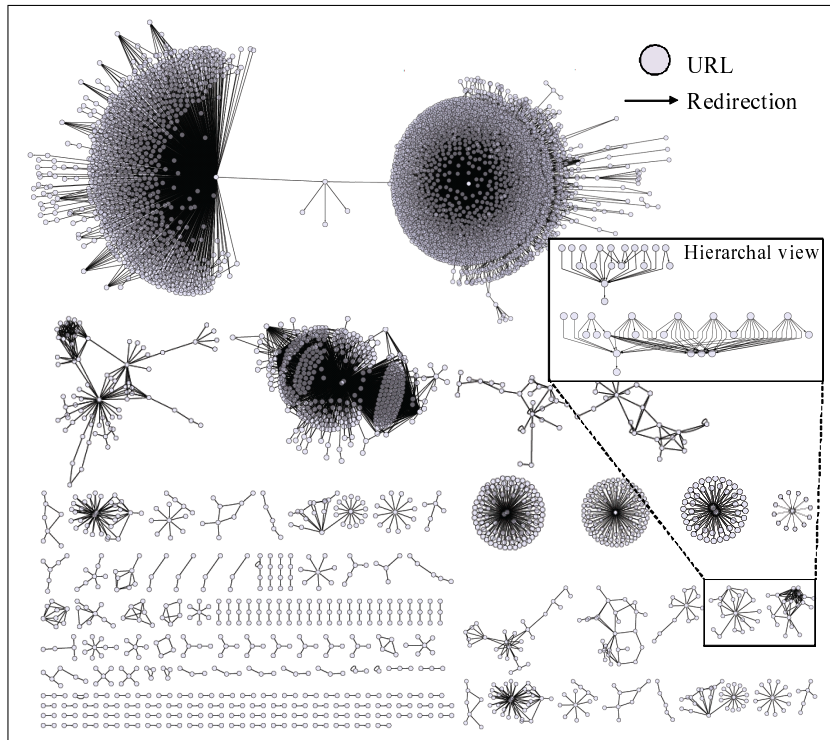
	Num
Execute	653
Communicate	399
TCP	389
HTTP	382
PORT_80	16
IRC	1
Other	8
UDP	291
DNS	287
Other	19
Non-Comm.	154
Non-Exec.	42

**Table 4.** Statistics for categories of redirect used by detected malicious sites

Category	Percentage
Iframe	71.2
Frame	11.3
META-refresh	0.01
HTTP-30x	16.2
Other (JavaScript or Unknown redirect)	0.01

**Extracting redirect chain** We extracted source-destination URL pairs (and additionally extracted those of IP address pairs) and the category of redirection within detected URLs in Sect. 4.2. In this way, 17% of FQDNs and 34% of IP address were newly extracted. In other words, these sites were behind the first accessed URLs.

Detected malicious sites often use multiple redirect and multi-hop redirect methods. Statistics for the categories of redirect are shown in Table 4. Almost all the redirect methods mainly use iframe.

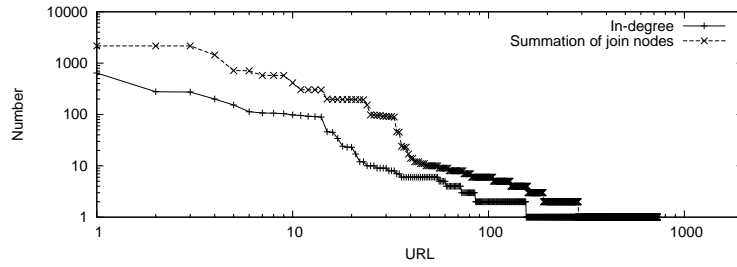


**Fig. 7.** Visualization of detected malware distribution networks

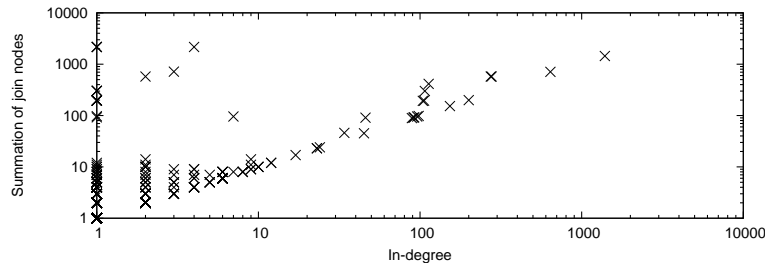
**Visualization** A visualization of a part of the observed malware distribution networks is shown in Fig. 7. Circles and arrows represent URLs and the direction of redirect, respectively. Most parts of the observed malware distribution networks represent a feature of scale-free networks. The basic structure is an abundance of landing sites chaining to several hub nodes (e.g., hopping sites or exploit sites). Some networks composed of thousands of nodes exist. The reason the number of unique binaries is only 1/40 of the number of collected malware binaries is that the same exploit sites were accessed and the same malware executables were downloaded in many cases.

We assume a representative countermeasure against web-based attacks is URL or IP Filtering. This countermeasure is effective for filtering hub nodes rather than filtering all nodes from these scale-free networks because the contents of hub nodes often include exploit codes and that of terminal nodes often include only redirect instructions. Moreover, because migration of nodes is difficult for large scale networks, the network structure of large groups is comparatively stable.

The number of passive references (in-degree) from each node and the number of join nodes which are underlying UPLs from certain URL to leaf URLs for redirection are shown in Fig. 8. The numbers are arranged from highest to lowest. A measure against nodes having many passive references (the left side of Fig. 8) would be effective. For example, if we filter the top ten URLs with regard to in-degree, we could defeat about 85% of detected malicious URLs without filtering each detected URL.



**Fig. 8.** In-degree and summation of join nodes under each URL by redirection



**Fig. 9.** Distribution of in-degree and summation of join nodes

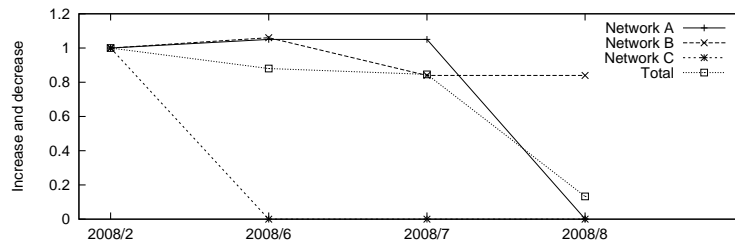
#### 4.6 Lifecycle of malware distribution networks

We checked the same sites on our URL list periodically for half a year. We observed the changing structure of malware distribution networks and the decrease of the malicious sites within. The increase and decrease of the number of sites in five of our chosen networks and the total number of detected URLs are shown in Fig. 10. This figure includes the top three large scale networks. As time advanced, the number of chain nodes gradually decreased for the majority.

Small scale networks and stand alone sites have the tendency to be comparatively unstable (i.e., to disappear within a short time or merge with other networks). In contrast, the structure of large networks is comparatively stable because node migration is difficult in large scale networks. For example, 91.6% of the nodes in the largest network (network A) had existed for at least 5 months. Meanwhile, small scale networks tend to survive shorter than that of large scale in our result. We assume that aggregated structure has trade-off between low management cost to high migration cost. For this reason, filtering of large scale networks is more effective than that of small scale networks or stand alone sites.

## 5 Discussion

Our detection method, in particular vulnerability-specific detection, covers only the specific version and kind of browser or plug-in. In contrast, the other detection methods of JavaScript-anomaly detection and local resource monitoring and controlling are basically independent of the browser version or kind. However, the system obviously cannot detect attacks without an exploit occurring at a specific version and kind of browser or plug-in. Thus, this false-negative issue is inevitable for a high interaction system in the case of attacks targeted at a specific version



**Fig. 10.** Increase and decrease of detected malicious sites for each network

or kind of web browser. Consequently, we should effectively adopt certain versions or kinds of web browser and BHOs to our system to portray an exploitable and attractive environment to adversaries.

How does our inspection cover a vast amount of web space? In Refs. [6] [7], the large web repository of a search engine is used for light-weight screening as the first step of web patrol. A combination of both putting target web contents through a sieve by a low interaction system and in-depth analysis by a high interaction system works effectively for large scale inspection.

How we collect the URLs for a seed URL list for investigation is important because the result of the investigation strongly depends on the list. In this research, we obtained some malicious URL lists in advance. Additionally, our system has a recursive inspection function that extracts hyper link URLs recursively from certain origin URLs to linked URLs. In consideration of public exposure, we should use keyword searches on famous search engines (e.g., Google, Yahoo, etc.). The modules of the systems in Refs. [5] [16] also have this function.

As mentioned in Sect. 4.6, malicious sites and their chains transform themselves in varying degrees. Moreover, predicting when a legitimate (however compromised) web server is changed to a malicious web site by an adversary's SQL injection attack is difficult. Thus, we should periodically inspect web servers and maintain the freshness of the collected information.

Detection methodologies of a client honeypot should cover the latest client-side attack targeting vulnerabilities. April 2009, JSredir-R was spread out on the web [9]. Typically, JSRedir-R is found on legitimate websites, hidden behind obfuscated JavaScript, loading malicious contents from third-party sites without the user's action. Malicious contents target adobe Flash Player's and Acrobat Reader's vulnerabilities. Marionette is already installed Adobe Flash Player and Acrobat Reader as BHOs, and able to detect above types of vulnerabilities by using newly implemented detectors against Flash Player's vulnerability (CVE-2008-2345) and Acrobat Reader vulnerabilities (CVE-2007-5659, CVE-2008-2992, CVE-2009-0658, CVE-2009-0927). Basically, Marionette is able to handle attacks targeting browser's plug-ins if specific plug-ins are installed.

## 6 Conclusion

We have presented the design and implementation of unique client honeypot, a high performance client honeypot able to accurately and circumstantially extract information of malicious web sites. Our proposed high interaction client honeypot called MARIONETTE enables the collection of various required information for an analysis of adversaries' methodologies and will eventually enable the development of countermeasures against web-based attacks and infection. In our continual investigation in field trials over the long term, we ascertained the methodologies

of adversaries and the stability of our proposed system. The result of investigations using MARIONETTE reveal the nature of web-based malware and the anomalous aggregated structure of malware distribution networks. We think our observation of the scale-free structure of malware distribution networks in the real field supports the development of an effective countermeasure.

## Acknowledgments

We would like to thank NTT Communications Corporation for supporting our research. This research was also supported in part by the Botnet survey research of the Ministry of Internal Affairs and Communications, Japan.

## References

1. Clam AntiVirus. ClamAV. <http://www.clamav.net/>.
2. Microsoft Research. Strider honeymonkey. <http://research.microsoft.com/HoneyMonkey>.
3. MITRE. Honeyclient project. <http://www.honeyclient.org/>.
4. Alexander Moshchuk, Tana Bragin, Steven D. gribble, and Henry M. Levy. A crawler-based study of spyware on the web. In *13th Annual Network and Distributed System Security Symposium (NDSS)*, 2006.
5. The Client Honeynet Project. Capture-HPC. <http://client-honey.net.org/>.
6. Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. All Your iFRAMES Point to Us. In *17th USENIX Security Symposium*, 2008.
7. Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, and Negendra Modadugu. The ghost in the browser: Analysis of web-based malware. In *the first USENIX workshop on hot topics in Botnets (HotBots'07)*, 2007.
8. SANS Internet Storm Center. SQL injection worm on the loose. <http://isc.sans.org/diary.html?storyid=4393>.
9. Sophos. Malicious jsredirect script found to be biggest malware threat on the web. <http://www.sophos.com/blogs/gc/g/2009/05/14/malicious-jsredirect-javascript-biggest-malware-threat-web>.
10. Symantec. Backdoor.sdbot. [http://www.symantec.com/security\\_response/writeup.jsp?docid=2002-051312-3628-99](http://www.symantec.com/security_response/writeup.jsp?docid=2002-051312-3628-99).
11. Symantec. Global internet security threat report volume XIII. <http://www.symantec.com/business/theme.jsp?themeid=threatreport>.
12. Symantec. Mpack, packed full of badness. [http://www.symantec.com/enterprise/security\\_response/weblog/2007/05/mpack\\_packed\\_full\\_of\\_badness.html](http://www.symantec.com/enterprise/security_response/weblog/2007/05/mpack_packed_full_of_badness.html).
13. Symantec. W32.mocbot.a. [http://www.symantec.com/security\\_response/writeup.jsp?docid=2005-102415-5716-99](http://www.symantec.com/security_response/writeup.jsp?docid=2005-102415-5716-99).
14. Symantec. W32.spybot.obb. [http://www.symantec.com/security\\_response/writeup.jsp?docid=2005-042211-4441-99](http://www.symantec.com/security_response/writeup.jsp?docid=2005-042211-4441-99).
15. The Client Honeynet Project. <http://www.client-honey.net.org/>.
16. The Client Honeynet Project. HoneyC. <https://projects.honey.net.org/honeyc>.
17. Yi-Min Wang, Doug Beck, Xuxian Jiang, Roussi Roussev, Chad Verbowski, Shuo Chen, and Sam King. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *13th Annual Network and Distributed System Security Symposium (NDSS)*, 2006.
18. Websense. Detecting web browser heap corruption attacks. <http://securitylabs.websense.com/content/Assets/BH2007-DetectingWebBrowserHeapCorruptionAttacks.pdf>.
19. Zero Emergency Response Team (ZERT). <http://isoft.org/zert>.