Int. J. Inf. Secur.
DOI 10.1007/s10207-017-0361-5

REGULAR CONTRIBUTION

HoneyCirculator: distributing credential honeytoken for introspection of web-based attack cycle

Mitsuaki Akiyama[1] · Takeshi Yagi[1] · Takeo Hariu[1] · Youki Kadobayashi[2]

Abstract A web user who falsely accesses a compromised website is usually redirected to an adversary's website and is forced to download malware after being exploited. Additionally, the adversary steals the user's credentials by using information-leaking malware. The adversary may also try to compromise public websites owned by individual users by impersonating the website administrator using the stolen credentials. These compromised websites then become landing sites for drive-by download malware infection. Identifying malicious websites using crawling techniques requires a large amount of resources and time. To monitor the web-based attack cycle for effective detection and prevention, we propose a monitoring system called HoneyCirculator based on a honeytoken, which actively leaks bait credentials and lures adversaries to our decoy server that behaves like a compromised web content management system. To recursively analyze attack phases on the web-based attack cycle, our proposed system involves collecting malware, distributing bait credentials, monitoring fraudulent access, and inspecting compromised web content. It can instantly discover unknown malicious entities without conducting large-scale web crawling because of the direct monitoring behind the compromised web content management system. Our proposed system enables continuous and stable monitoring for about one year. In addition, almost all the malicious websites we discovered had not been previously registered in public blacklists.

Keywords Web-based malware · Client honeypot · Malware sandbox · Honeytokens · Information leakage

✉ Mitsuaki Akiyama
akiyama.mitsuaki@lab.ntt.co.jp

Takeshi Yagi
yagi.takeshi@lab.ntt.co.jp

Takeo Hariu
hariu.takeo@lab.ntt.co.jp

Youki Kadobayashi
youki-k@is.naist.jp

1 NTT Secure Platform Laboratories, NTT Corporation, 3-9-11 Midori-cho, Musashino-shi, Tokyo, Japan

2 Nara Institute of Science and Technology, Takayama 8916-5, Ikoma, Nara, Japan

1 Introduction

Attacks by *Beladen*, *Gumblar*, and *Nineball* are large-scale incidents of mass compromises of websites [1]. These types of compromises are the leading cause of malware infection on the web. Web content on a compromised website is injected with code that will redirect web clients unknowingly to a malicious website. The malicious website behind the compromised website that acts as a *landing site* contains exploit code that targets the web browser's vulnerability. The web client is automatically infected with malware simply by accessing the compromised website without user interaction. If the malware has the functionality of leaking information, the credentials on a victim host may be unknowingly stolen. Furthermore, if a victim host stores its own website credentials, the credentials can also be stolen and leaked to adversaries. Adversaries can then compromise that website with the stolen credentials (Fig. 1). In this way, the phases of an attack cycle, such as *exploit*, *credential collection*, and *fraud & abuse*, are repeated continuously, further spreading malware infection and compromising more websites. In this attack cycle, malicious entities, such as specific exploit codes and exploit websites behind compromised websites, might change temporarily, and compromised websites might also become other secondary attack vectors. We believe it is

Published online: 30 January 2017

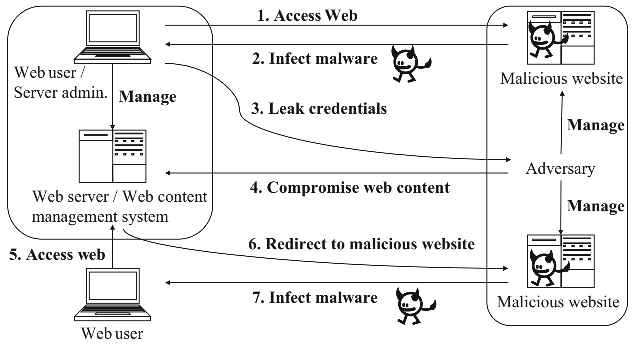

Fig. 1 Attack model. Exploit, credential collection, and fraud and abuse phases represent 1–2, 3, and 4–7, respectively

important to comprehensively monitor an adversary's activities in this attack cycle to understand the above-mentioned temporal changes and secondary attack vectors [2].

Much research has been done to find ways of discovering malicious websites in web space [3–6]. The typical approach is crawling-based malicious website discovery. This approach, however, requires numerous system and network resources for inspection, so it is a time-consuming process. Effective methods for discovering suitable seed URLs to apply crawling have been proposed recently [7–9]. Although these guided crawling methods can dramatically reduce the cost of crawling, another problem still remains, i.e., the need to rapidly discover unknown malicious websites. Moreover, in client-side monitoring (i.e., crawling as a web browser), it is impossible to understand how adversaries use stolen credentials or to understand the activities of adversaries on compromised servers.

One kind of honeypot is a *honeytoken*, which is artificial digital data planted deliberately into a genuine system resource in order to detect unauthorized attempts to use information [10]. In actual deployments, a honeytoken is leaked to an adversary as a bait credential, such as an account name or password, in order to monitor malicious usage of actual servers or services.

To consistently monitor such web-based attack cycles, we propose a monitoring system called HoneyCirculator based on a honeytoken that deliberately leaks bait credentials and lures adversaries to our decoy server that behaves like a compromised web content management system. Our assumption is that honeytoken-based monitoring can be used for in-depth tracking of a series of the above-mentioned attack phases by a specific adversary and can discover malicious entities (e.g., adversary's IP addresses, malicious websites, exploit codes, redirect codes). The contributions of this paper are as follows:

– We established a monitoring system that succeeded in being continuously compromised by various adversary infrastructures in a one-year experiment.
– Our proposed system discloses adversaries' activities on compromised websites: traffic direction to exploit websites, web access control to circumvent security inspection, phishing-based credential exfiltration, and mail-based drive-by download.
– We verified through a field experiment that the monitored information instantly revealed unknown malicious IP addresses and domains without needing to conduct large-scale web crawling when they are used, and most of the information was not contained in public blacklists.

The remainder of this paper is organized as follows. Related work is described in Sect. 2, and assumed malware infection and a preliminary investigation of information-leaking malware are described in Sect. 3. Our honeytoken-based monitoring system is explained in Sect. 4, and the results of experiments conducted to obtain information on adversaries' activities in actual web space and an evaluation of the monitored malicious entities are presented in Sects. 5 and 6, respectively. Discussion and concluding remarks are given in Sects. 7 and 8, respectively.

2 Related work

2.1 Honeytoken

The first definition of a honeytoken was given by Spitzner [10]. A honeypot can also be a piece of digital information, which is a special form of honeypot called a honeytoken. It can be an office document, e-mail address, database entry, or login credential that serves as bait to lure adversaries. A unique advantage of honeytokens is monitoring the flow of stolen information and its abuse.

HoneyGen [11] is a method for automatically generating honeytokens that are similar to actual data by extrapolating the characteristics and properties of actual data items that are difficult to distinguish from actual data.

There are two types of honeytoken distributions: *victim-trigger* and *adversary-trigger*. Victim-trigger-based leakage is when a victim intentionally submits honeytokens (e.g., credentials) to a malicious server such as a phishing site. Monitoring systems of victim-trigger-based honeytoken leakage have also been developed to detect *phishers* by submitting credentials to phishing sites and monitoring their usage [12–14]. Adversary-trigger-based leakage is when an adversary or malware intrudes into a victim's system and steals honeytokens. The D^3 [15] system automatically generates decoy documents with credentials and prompts an adversary to exfiltrate them. BotSwindler [16], which is a hybrid leakage method, emulates user interaction, e.g., keyboard input, on a malware sandbox and prompts malware with a trigger condition of user interaction to leak credentials of actual services and monitors the use of stolen credentials on these services to identify information-leaking malware.

Many monitoring models are basically designed to involve genuine authoritative services (e.g., credentials of banking, webmail, and shopping services). Compromised accounts on other services are beyond the control of the honeytoken system; therefore, they potentially exhibit serious ethical and legal problems. Our proposed system uses adversary-trigger-based leakage and can precisely track complex attack cycles without actual victims because all components in our monitoring are decoys.

2.2 Collecting and analyzing exploit codes and malware

Glastopf [17] emulates vulnerable web applications and receives exploit codes targeting them. Canali's work [18] focused on the behavior of adversaries after a website has been compromised and clarified what adversaries do behind exploits, so the motivation is similar to that of ours. Canali et al. deploy vulnerable content management systems for monitoring and classifying patterns of adversary behavior. The attack vector expected with these systems is vulnerability of web applications. Due to the large amount of flexibility for adversaries to intrude, it successfully monitored various types of adversary behavior. In contrast, HoneyCirculator is only focused on the intrusion based on stolen credentials, i.e., *masquerade attack*. While there is not a large amount of flexibility for adversaries on our web content management system (WCMS) honeypot, it is easy to prompt adversaries to compromise web content leading to drive-by download exploits. In addition, the most different is *recursively* analyzing malicious entities on the web-based attack cycle.

Online sandbox services [19,20] collect and analyze malicious executables and URLs submitted by globally distributed users. *ShadowServer* [21] analyzes malware (bots) provided by collaborative organizations to extract command and control (C&C) servers and continuously tracks botnet activity via these servers. Kreibich et al. proposed a sandbox farm called GQ [22] that controls fine-grained C&C communications to maintain safety and allows flexible/precise containment policies.

3 Assumption and preliminary investigation

Many malicious websites targeting browser vulnerabilities have recently appeared. When a vulnerable web client accesses such websites, the exploited web client is unknowingly forced to download/install malware without user interaction. This type of exploitation is called *drive-by download*. Because drive-by downloads are executed in accordance with legitimate web protocols (i.e., HTTP and HTTPS), port-blocking- or protocol-anomaly-based detection methods are not effective countermeasures. Because various Internet services have been integrated into the web in the last decade, the functionality of web browsers has been enhanced by various plug-in applications. This enrichment of browser functionality unfortunately results in a situation in which the vulnerabilities of different software vendors are continuously exposed, which requires patch management by software vendors to repair such vulnerabilities. However, by the time some patches have been released, some web browsers will have already been exploited. For these reasons, i.e., the use of legitimate protocols and the patch management problem, drive-by download is now becoming the main malware infection vector.

Malicious websites that attempt to perform drive-by downloads lure general public web clients to their websites using various techniques: with the link-URL of spam e-mail, search engine optimization, and by compromising benign websites so they serve as landing sites of backend malicious websites. This paper focuses on malware infection that uses drive-by downloads and also lures web clients to compromised landing websites that redirect them to backend malicious sites. When a vulnerable web client accesses a compromised website, it is redirected to a backend malicious website containing exploit code, and as a result, is infected with malware. Moreover, if the infected host has credentials, the malware steals them and sends them to a remote host owned by an adversary. An adversary can then access and compromise web content by using the stolen credentials. Compromised web content that has been injected with redirect code leads other web clients to backend malicious sites. Thus, this type of malware spreads like a chain reaction.

We first needed to identify what kinds of applications were being targeted for credential theft in order to design an analytical environment that would allow credentials to be stolen. Our preliminary investigation involved analyzing the internal behavior of malware on an infected system (e.g., filesystem and registry access) in order to identify what applications information-leaking malware targets. We collected malware executables from a public blacklist (MalwareDomainList [23]) and analyzed them. We confirmed from the results of malware analysis that various malware executables attempted to access specific files and registries corresponding to applications of mail clients, web browsers, file managers, web authoring tools, instant messaging clients, and file transfer protocol (FTP) clients. These enumerated applications seem to be *potential* victims of information leakage.

Particularly, various malware executables simultaneously accessed files and registries of various FTP clients. Accessed files and registries store configuration information of certain applications, including credentials (account name, password, IP address/fully qualified domain name (FQDN) of server), which are written in plain text or encoded using a simple encoding algorithm, e.g., XOR or bit shifts. An adversary can

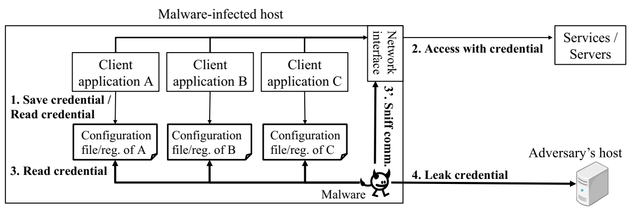

Fig. 2 Extracting and leaking credentials on victim host

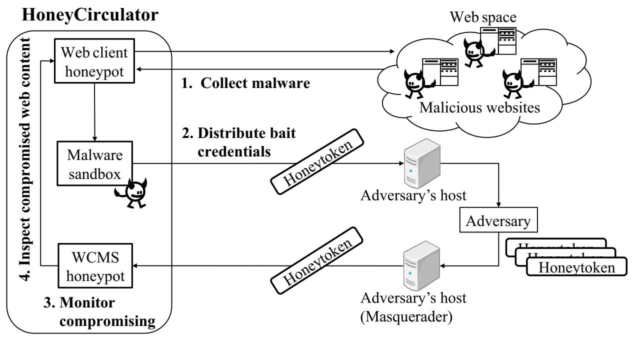

Fig. 3 Overview and analytical procedure of HoneyCirculator for monitoring attack cycle based on credential leakage

easily decode them if he or she knows what the encoding algorithms client applications use. In addition, when an FTP client accesses an FTP server, malware executables sniff that communication between them and extract credentials in some cases. Malware sends the stolen credentials to a remote host owned by an adversary. How information-leaking malware extracts and leak credentials on a victim host is illustrated in Fig. 2. It has been reported that incidents of mass compromising, e.g., Gumblar/Beladen/NineBall introduced in Sect. 1 result from leakages of FTP credential by malware [24]. Although FTP is a simple and conventional service, it is still widely used for managing files on servers; consequently, FTP is still a target for masquerade attacks and mass compromises.

In addition, we found evidence of FTP credential leakage in which there are credentials related to FTP accounts in plain text or the Base64 encoding format in the payloads of communication between the malware and remote host. Over 30 kinds of FTP client applications were potentially compromised by malware in the preliminary investigation. Therefore, we believe that FTP clients are the most targeted applications for credential leakage.

4 Design of monitoring system

4.1 Analytical procedure

The primary aim with HoneyCirculator is monitoring an adversary's behavior at each attack phase, seamlessly tracking the phases, and clarifying the complex attack cycle. The analytical procedure was conducted as explained in the following steps (also shown in Fig. 3) using three key components: a web client honeypot, malware sandbox, and WCMS honeypot. Honeytokens transparently come and go between our malware sandbox, adversaries, and our WCMS honeypot. These components are described briefly and in more detail later in this section. The analytical procedure, which is repeatedly executed, is described as follows.

- Collecting malware
  Our web client honeypot crawls malicious websites listed in the latest blacklist and collects the latest malware executables. The collected malware executables are sent to the malware sandbox.
- Distributing bait credentials
  Our malware sandbox analyzes the collected malware executables. In each analysis, the malware sandbox randomly generates unique credentials. If an analyzed malware executable has information-leaking functionality, it exfiltrates information, e.g., credentials, to a remote adversary.
- Monitoring fraudulent access and compromised web content
  Our WCMS honeypot actually behaves as an FTP server. It creates a user directory for each account corresponding to potentially leaked credential and awaits a masquerade attack with stolen credentials. It stores login, command, and file histories in each account, while a masquerader intrudes on it then operates files that are web content.
- Inspecting compromised web content
  The web content on our WCMS honeypot, which is compromised by a masquerader, is assumed to be injected with redirect code leading to malicious websites for drive-by downloads. Our web client honeypot inspects the web content and collects information on unknown malicious websites as redirect destinations.

4.2 Building blocks

We designed HoneyCirculator as an automatic monitoring system composed of the three aforementioned components: a web client honeypot for collecting malware executables, a malware sandbox for analyzing malware and distributing credentials, and a WCMS honeypot for monitoring malicious access. Additionally, a credential honeytoken acts as a bait credential of a WCMS and chains the above monitoring components.

4.2.1 Web client honeypot

A web client honeypot is a decoy web browser for detecting web-based malware infection and discovering malicious websites corresponding to web-based malware infection.

We use a web client honeypot we previously developed [25,26] to collect malware executables from the web. Our web client honeypot has three characteristics: running on a high-interaction system, tracking malicious websites chained by redirection, and collecting malware from both drive-by download and click-download.

There are two types of honeypots, depending on their *interaction* with websites: *high-interaction* using an actual system [25,27,28] and *low-interaction* using an emulator [29,30]. With drive-by download attacks, an exploit code targeting various types of vulnerabilities is contained in the web content processed using many client applications (e.g., web browser, Acrobat, Java, Flash). Moreover, web content that includes an exploit code is often obfuscated; the exploit code appears when a web browser processes the web content. Therefore, the detection accuracy of a low-interaction system strongly depends on how faithfully an emulator simulates client applications. By contrast, high-interaction web client honeypots use an actual system, so exploitation is generally successful when a targeted vulnerability exists on a system. This type of honeypot can detect exploitation from anomalous system behavior. A representative high-interaction detection method involves monitoring filesystem/registry access events and process-creation events and validating that these events follow predefined behavior rules [27]. Our web client honeypot is also high-interaction and installed a vulnerable version of Internet Explorer and plug-ins such as Flash player, Adobe Reader, and Java.

Tracking a *malware distribution network* (MDN) composed of malicious websites is important for identifying the backend core sites of drive-by download attacks. HoneyMonkeys [28] can analyze URL redirection based on redirection of the HTTP protocol, HTML tags, and JavaScript. Our web client honeypot can precisely track MDNs by analyzing the redirect relationship of malicious websites and parsing HTTP query/responses and extracting link URLs from the browser's DOM tree.

Another type of web-based malware infection is a *click download*, which is when a web user accesses a URL that points directly to an executable file (e.g., http://example.com/malware.exe) and clicks on the download dialog. Our web client honeypot can handle the dialog by emulating a click event and downloading the executable file. Although click-downloaded executables are not always malware, our web client honeypot collects all click-download executables and labels them as "click download malware," as described in Sect. 5.

4.2.2 Malware sandbox

To prompt a malware executable to leak credentials, we need to run it in an environment with Internet access. An execution environment for malware analysis is called a *malware sandbox*. A malware sandbox is usually managed with no Internet access in order to block attacks to remote hosts or networks (e.g., denial of service, scanning, mass-mailing, remote exploit attacks). However, a bot or download-type malware communicates with a remote host that is a C&C server, and information-leaking malware also communicates with a remote host in order to send stolen credentials. Therefore, a malware sandbox requires Internet accessibility to analyze information-leaking malware. As a safeguard, our malware sandbox [31] provides semi-permeable Internet accessibility in which it permits only DNS and HTTP communication including C&C communication. It redirects other communication, which is assumed to be attack activities (e.g., SMTP for mass-mailing, TCP139/445 for remote exploits), to internal fake servers.

Our malware sandbox is composed of a sandbox agent, which is a victim OS running malware, and *GateKeeper*, which controls the malware's communication. GateKeeper intercepts communication from a sandbox agent and identifies the layer-4 protocol (i.e., TCP/UDP) and upper layer protocols and delegates processing to the appropriate *FakeDaemon*, which acts as a proxy server and handles upper layer protocols (e.g., HTTP-FakeDaemon) in a virtual network in the sandbox environment. If a certain communication protocol is permitted, a corresponding FakeDaemon passes that communication through the Internet. If not permitted, a corresponding FakeDaemon responds to the sandbox agent as a fake response. In our experiment, we set HTTP/IRC/DNS as the protocols permitted to send outside.

As mentioned in Sect. 3, we analyzed information-leaking malware as part of our preliminary investigation and discovered that many kinds of FTP client applications were being targeted. Therefore, we used FTP credentials as honeytokens on our sandbox. How to set up honeytokens is described in 4.2.3.

4.2.3 Credential honeytoken

A honeytoken deployed by HoneyCirculator is a bait credential of a remote FTP server. Therefore, it requires either the remote server's IP address or FQDN, or both as well as an account and password pair. Based on this assumption, a honeytoken $T = (A, P, I, D)$ is defined as the tuple of A, P, I, and D, where A is the account name, P is the password, and I and D are the IP address and FQDN of our aforementioned WCMS honeypot, respectively. Our malware sandbox produces a unique honeytoken (T) in each malware analysis.

A honeytoken as a credential should have three properties: 1) be able to be identified by the honeytoken generator, 2) be difficult to be identified as a bait credential by an adversary, and 3) be complex enough to be unpredictable for the adversary in order to discriminate a fraudulent access with stolen credentials from a fraudulent access based on password brute force or password list attack. The *A* and *P* of the honeytoken in each analysis are randomly generated alphanumeric strings based on the above properties. The relationship between a certain *T* and the analysis has a one-to-one correspondence.

On the sandbox, an account and password pair are generated as parts of the honeytoken in each analysis; then, they are stored in a configuration file or registry before running a malware executable. They are followed the same format and encoding as those of actual client applications. They are also simultaneously generated on the sandbox in order to increase the chance of credential leakage. Based on our preliminary investigation described in Sect. 3, we selected nine kinds of client applications frequently targeted by credential leakage. In addition, after starting analysis, our sandbox generates FTP communication to a dummy FTP server in consideration of other methods of stealing credentials such as sniffing communication.

Our system does not require distinguishing to where credentials are leaked, but it can definitely identify leaked credentials when they are used by a masquerade attack. To recognize the relationship between the account name of *T* and the malware analyzed by the malware sandbox, we can coordinate the malware sandbox log and WCMS honeypot log as follows.

4.2.4 WCMS honeypot

Our WCMS honeypot shown in Fig. 4 is a server-type honeypot that acts as a file server of a website in order to capture compromised web content and monitor an adversary's activity on the server. It is assigned a specific domain name because adversaries access it by using either the server's IP address or domain name described in the stolen credentials.

Only an FTP daemon runs on our WCMS honeypot, and other network services (e.g., web server) are basically stopped for legal or ethical issues because we reduce the risk that general public web users may falsely access our compromised web content.

Generally, there are two patterns of fraudulent access attempts: masquerade attack with stolen credentials and brute-force attack. In a masquerade attack, login attempts with stolen credentials are always successful. Although a brute-force attack requires many login attempts, such attempts to login to our WCMS honeypot have never been successful because the username/password pairs are randomly generated as alphanumeric strings (as mentioned in Sect. 4.2.3), so they are sufficiently complex to thwart such brute-force attacks. Our WCMS honeypot has no genuine account; therefore, it can determine that all login attempts are not legitimate. In addition, based on the list of our generated honeytoken, it can precisely identify masquerade attacks with stolen credentials and enumerate masqueraders' IP addresses in a timely manner.

To camouflage a genuine system, our WCMS honeypot prepares bogus web content (e.g., `html`, `php`, `js` files) for each user directory of the FTP account and permits FTP users to access their own directory. Our WCMS honeypot stores FTP login, command, and file histories in each account. When an original file is changed, our WCMS honeypot preserves the changed file.

When web content is compromised, it must be inspected by our client honeypot. To be inspected, our WCMS honeypot temporarily runs a web server that is accessible only inside the internal network and enables the content to be inspected only by our client honeypot mentioned in Sect. 4.2.1.

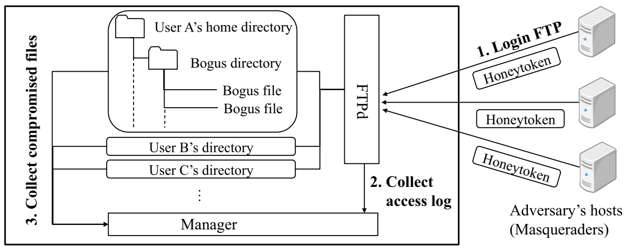

Fig. 4 WCMS honeypot for monitoring adversary's activity on compromised web content management system

5 Field experiment

Our objective in developing this system was to understand adversary activity on a compromised server and to effectively discover malicious websites conducting drive-by download attacks. By doing so, we can supply the IP addresses of adversaries trying to compromise the server and also supply malicious websites to security vendors and other potential victims in order for them to apply proactive countermeasures, such as blacklisting or filtering, before the adversaries are actually able to use the IP addresses and websites. This system was evaluated from March 2012 to February 2013. In this section, we uncover several aspects of malicious activities based on distinct adversary infrastructures, property and lifespan of infrastructure, targeted client applications, server-side behavior, and compromised web content.

5.1 Malware collection

Our experiment involved seed URLs for crawling in order to collect malware. In each crawling, we used the latest version of a public blacklist (MalwareDomainList [23]) containing about 80,000 URLs. We also used about 150,000 URLs of personal and commercial websites retrieved from search engines. Crawling was conducted over two- or three-day intervals. Our web client honeypot also inspected compro-

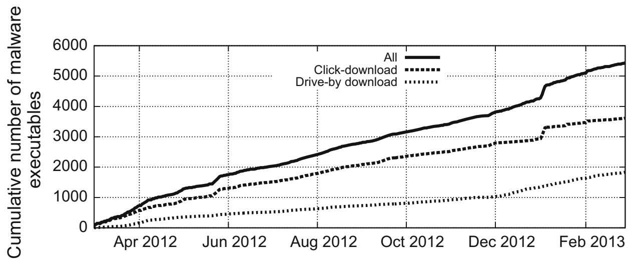

Fig. 5 Collected malware (unique hashes, collected from March 2012 to February 2013)

mised web content injected with redirect code on our WCMS honeypot and collected malware from a malicious website as a redirect destination. It collected malware executables via drive-by download and click-download (Fig. 5). Specifically, our client honeypot crawled about 53.4 million input URLs (5.6 million unique input URLs) and then collected a total of 5439 malware executables: 1833 by drive-by download and 3614 by click-download (8 were obtained both ways).

As shown in Fig. 5, the number of collected unique executables increased as time progressed. This means that our web client honeypot *stably* collected unique malware executables over the long term. The reasons for this are: (1) seed URLs (blacklisted URLs and compromised content URLs on our WCMS honeypot) were updated with each crawling, resulting in the capture of more executables from new malicious websites and (2) different executables were collected from known malicious websites; in other words, adversaries updated the executables they intended to distribute. Collected malware executables are analyzed by our malware sandbox within 24 h of collection. Even if our web client honeypot collected the same malware executables in different crawling actions, our malware sandbox analyzes them in each case because collected executables must be active at that time.

5.2 Compromised accounts

Our monitoring procedure can identify leaked credentials when they are actually used. At the beginning of the malware collection in March 2012, we established the experimental settings for our WCMS honeypot, e.g., the IP address assignment and domain name registration. Actual compromising started on April 5. Figure 6 indicates the time between leakage of a credential and first use of it. Adversaries try to access an account after a certain incubation period. Although 13.2% (44/332) of leaked accounts was initially accessed within 24 hours, many other accounts took several days to be initially accessed.

We can identify an adversary's IP address that controls information-leaking malware by monitoring accesses from remote FTP clients to our WCMS honeypot. To discriminate accesses using leaked credentials or brute-force accesses, we

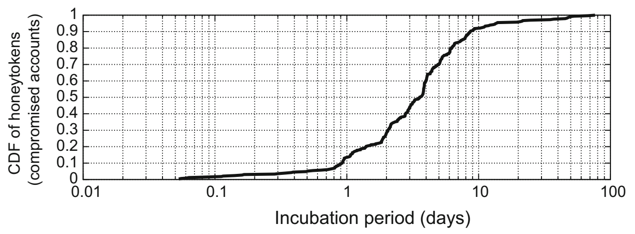

Fig. 6 Incubation period of website compromising: time between leaking and using credential

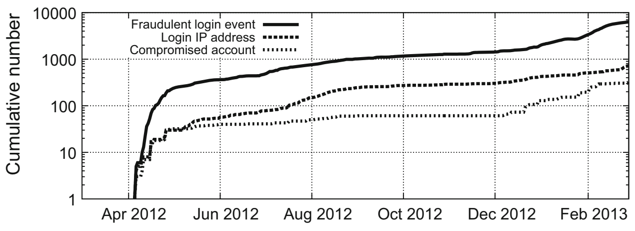

Fig. 7 Cumulative number of login events, login IP addresses, and compromised accounts

extract accesses obtained with a correct account name and password pair and exclude other false accesses. The cumulative numbers of adversaries' IP addresses and accounts that were successfully accessed are shown in Fig. 7. We monitored 332 compromised accounts, 6320 fraudulent login events, and 722 IP addresses of adversaries.

5.3 Adversary infrastructure

Monitored attacks are not always conducted by a single adversary, and there must be various attack campaigns controlled by various adversary infrastructures. From our WCMS honeypot log, we recognized many-to-many associations between masquerader's IP addresses and compromised accounts, for example, specific masquerader accesses many accounts and vice versa. We assume that the masquerader hosts are infected with bot and some adversaries conduct masquerade attacks using the sets of stolen credentials via the masquerader hosts. To analyze our large amount of monitored data and reveal the properties of each adversary infrastructure, we try to classify the basic components of an adversary infrastructure, i.e., masquerader's IP addresses and stolen credentials, into groups based on the graph structure of their associations. In this section, we explain the graph partitioning process to appropriately divide graphs into adversary infrastructures.

We defined an adversary's activity graph as $G = (v, e)$, where $v(G)$ are the entities of a masquerader's IP addresses and compromised accounts (stolen credentials), and $e(G)$ are their associations. The login history on our WCMS honeypot recorded 6320 fraudulent login events including 722 IP addresses and 332 accounts. With the above simple method, 14 graphs were produced and the largest graph had 88.4% of all vertices (639 IP addresses and 293 accounts). In regard to the largest graph, we assumed that multiple infected hosts (IP addresses) controlled by different adversaries resulted in the merging of some subgraphs. In our manual analysis, we confirmed that this giant graph had some weak edges bridging subgraphs. Therefore, our graph partitioning process should cut weak edges from multiple affiliated hosts in the largest subgraph against the above-mentioned over-merging problem. We show the grouping process of adversaries in Fig. 8.

5.3.1 Graph partitioning

Clustering algorithms generally exhibit the following two problems: they require features which usually depend on domain-specific heuristics, and they are NP-hard in both theory and in practice where the problem is relaxed to local optimization [32]. To solve these problems, we adopt *spectral graph partitioning*, which is a simple mathematical method based on basic liner algebra. It is currently a better solution to solve these problems so that many researchers can use it [33, 34]. The method requires only a feature based on the graph structure without domain-specific features and can solve the relaxation of an NP-hard problem [32]. Specifically, spectral graph partitioning is based on the graph Laplacian and used for cutting the weak connection (i.e., edges) of a graph. The graph Laplacian L, which is $n \times n$ matrix (n is the number of vertices), is the difference between the degree matrix D and the adjacency matrix A of the graph, which is defined as

$$L = D - A$$

$$(i, j)th \text{ component of } L = \begin{cases} \text{degree}(v_i) & i = j \\ -1 & i \neq j \text{ and } e(i, j) \in G \\ 0 & \text{otherwise} \end{cases}$$

Note that *i* or *j* stands for a node-ID that is uniquely assigned from 1 to n. Spectral graph partitioning [35,36] is a graph partitioning method based on eigenvectors of the graph Laplacian. It focuses on the *Fiedler vector*, which is the eigenvector associated with the second smallest eigenvalue of the graph Laplacian. The Fiedler vector's *N*th component corresponds to the *N*th node (node-ID). This eigenvector is called the *algebraic connectivity* of a graph. Clustering nodes based on Fiedler vector component values stand for cutting weak connections between vertices (i.e., edges) in a graph. We made a graph Laplacian of the largest subgraph and calculated its Fiedler vector. The results shown in Fig. 9 indicate three obvious levels of values corresponding to nodes. We used *gap cut* [36] detecting max $|v_n - v_{n+1}|$ between the largest gap in the sorted list of the Fiedler vector's components in order to cut the graph.

The two obviously large gaps were cut in our partitioning process, and the result of graph partitioning is illustrated in Fig. 10. The largest subgraph was partitioned into three subgraphs marked as groups A, B, and D. The set of IP addresses in the same subgraph is defined as an adversary infrastructure, and we monitored 16 such adversary infrastructures in our experiment.

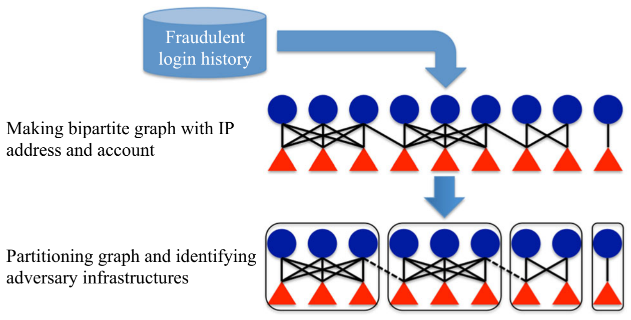
Fraudulent login history
Making bipartite graph with IP address and account
Partitioning graph and identifying adversary infrastructures

Fig. 8 Grouping process of adversary infrastructure. *Blue circles* and *red triangles* indicate masquerader's IP addresses and compromised FTP accounts, respectively (color figure online)

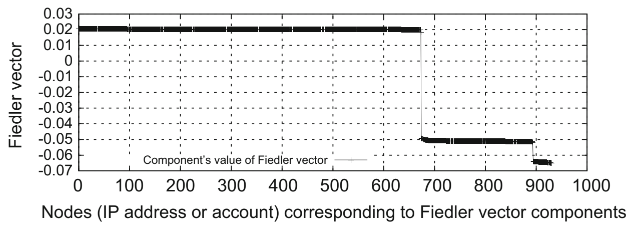

Fig. 9 Fiedler vector of graph Laplacian. Fiedler vector component values are arranged from largest to smallest. Two obviously large gaps are candidates for partitioning point

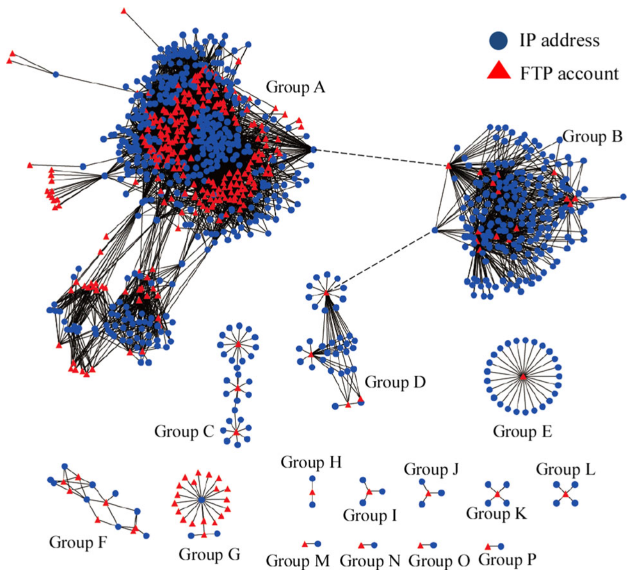

Fig. 10 Partitioned activity graph of adversary. Each *graph* indicates an adversary infrastructure. Two edges, depicted as *dashed lines* in this figure, are cut in our graph partitioning process

Table 1 Properties of adversary infrastructures

Adversary infrastructure	# of adversary's IP addresses	# of compromised accounts	# of malware executables	# of fraudulent login events
Group A	401	273	168	4921
Group B	205	15	15	803
Group C	26	3	3	28
Group D	33	4	4	215
Group E	25	1	1	91
Group F	9	7	6	68
Group G	3	19	18	149
Group H	2	1	1	2
Group I	3	1	1	4
Group J	4	1	1	4
Group K	4	1	1	20
Group L	4	1	1	11
Group M	1	1	1	1
Group N	1	1	1	1
Group O	1	1	1	1
Group P	1	1	1	1
Total	722	331	224	6320

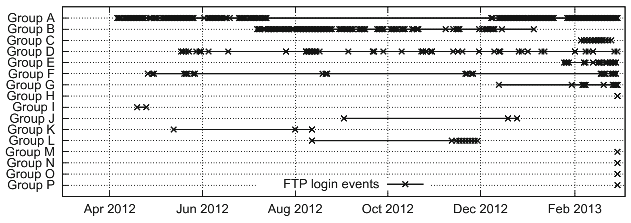

Fig. 11 Lifespans and activities of adversary infrastructures

5.3.2 Property of adversary infrastructure

There are several types of cardinalities in which a single IP address accesses many FTP accounts or many IP addresses access a single FTP account. These clusters indicate the operational characteristics of each adversary infrastructure. An adversary collects many credentials and accesses their accounts via various bots. We assume that an adversary uses bots and accesses common FTP accounts using their IP addresses in large clusters. We confirmed that the activities that each infrastructure carried out to compromise web content were similar within the infrastructure but differed between infrastructures.

The properties of the adversary infrastructures are listed in Table 1, and Fig. 11 shows their lifespans and activities based on our monitoring. Groups A and B had a particularly large number of compromised accounts. Groups A, C, D, E, F, and G continued to operate for some months. Activities of groups B, I, J, K, and L stopped for several months. Groups H, M, N, O, and P started operating at the end of our monitoring period in February 2013. These groups might have integrated into the same group if we had monitored them longer. Our server-side monitoring focusing on each infrastructure enables us to clearly understand when a group starts its operations and when it stops.

The most remarkable finding is the complementarity of activity between groups A and B, which are divided using the previous graph partitioning process. In the period of stopping group A's activity, group B starts its operation. After group B is low in activity, group A restarts its operation. Based on the tendency of activity, it indicates that a specific adversary conducts three attack campaigns by taking on different infrastructures. This finding emerges only from focusing on the timelines of each infrastructure.

5.4 Malware leaking information

We identified malware executables that leaked credentials. Our malware sandbox stores pairs of malware executables and generated credentials in each analysis. When a specific account is accessed, we can find a malware executable corresponding to the honeytoken including that account from the malware sandbox log. Three hundred and seventy-three out of 5439 malware executables had obviously accessed honeytokens, i.e., our prepared configuration files/registries of FTP clients, and had sent suspicious HTTP requests outside, and 224 out of them had positively leaked. Therefore, although 6.8% of the collected web-based malware probably leaked credentials, 4.1% successfully *reused* them in our assumed attack cycle. The honeytokens leaked by 2.7% of malware

Table 2 Information-leaking malware families (total of 224 executables)

Malware family	#
A. McAfee	
Generic BackDoor.*	81
PWS-Zbot*	52
BackDoor-FJW!*	51
Other malware	38
Unknown	2
B. Kaspersky	
Trojan-PSW.Win32.Tepfer.*	69
Trojan-Downloader.Win32.Agent.*	68
Trojan.Win32.Bublik.*	14
Other malware	49
Unknown	24
C. Symantec	
W32.Waledac.D*	83
Trojan.Gen*	47
SecShieldFraud*	27
Other malware	44
Unknown	23

were not reused for certain reasons; for example, our system was exposed as a decoy. Of the 224 executables, 15 were collected by click-download and 209 by drive-by download. The scanning results of three antivirus applications (*McAfee*, *Kaspersky*, and *Symantec*) with the latest pattern files are listed in Table 2. The *Zbot* and *Tepfer* families are known as *information stealers* such as of banking information and other credentials. *Waledac* and *Kelihos* are associated with credential theft [37] and were detected as *Win32.Waledac* by Symantec. Although some executables categorized as *generic* or *unknown* mean that antivirus applications cannot explicitly identify their family names, security venders' nomenclatures of malware are reasonable based on our monitoring results of fraudulent access.

5.5 Credential leakage behavior

5.5.1 Internal behavior on infected host

Our sandbox prepares configuration files/registries of nine kinds of clients (A–F including four different versions of A). What kinds of client tend to be stolen credentials and how many clients malware executables target simultaneously are listed in Table 3. Clients of A_1–A_4 and E are targeted by about three-quarters or more of information-leaking malware. Although 23 malware executables did not access configuration information of clients, 6 malware executables out of 23 communicated with remote hosts after FTP communication generated by our sandbox. Therefore,

Table 3 Clients targeted by information-leaking malware

A. Clients and targeting malware Targeted clients	# of malware executables
A_1	190 (84.82%)
A_2	179 (79.91%)
A_3	190 (84.82%)
A_4	190 (84.82%)
B	27 (12.05%)
C	27 (12.05%)
D	23 (10.26%)
E	163 (72.76%)
F	20 (8.92%)
None	23 (10.26%)
B. Combination of targeted clients # of targeted clients	**# of malware executables**
0	23 (10.26%)
1	5 (2.23%)
2	2 (0.89%)
3	1 (0.44%)
4	35 (15.62%)
5	136 (60.71%)
6	3 (1.33%)
7	2 (0.89%)
8	8 (3.57%)
9	9 (4.01%)

at least six executables' method of leaking credentials seems to involve sniffing FTP communication. The method of the rest of the executables (17 executables) is unknown. About 90% of malware simultaneously leaked credentials of various clients, and about 75% targeted 4 or 5 clients, which were also A_1–A_4 or E.

5.5.2 Leakage communication

After extracting credentials, malware executables must send them to remote hosts owned by adversaries. In this experiment, malware executables leaking credentials sent HTTP-GET or -POST requests, both of which include an unreadable message body that seemed to be encrypted at least once in each analysis. It seems that stolen credentials were in it. Although including a message body with a GET request is not defined as standardization of HTTP, a web server can actually receive the request and extract it.

5.6 Adversary behavior on compromised server

Adversary behavior after intruding is one of the concerns of security researchers in understanding their intention. Adversaries can only use FTP commands on our WCMS honeypot because they log into it as FTP users. We analyzed adversary behavior, i.e., command sequences on their FTP sessions, and classified their command sequences into the following four representative types.

A. *File creation* An adversary creates a new file. This behavior is defined as only the command; "`STOR 'fileA'`," where *fileA* is a new file and has never been retrieved before in a session (i.e., "`RETR 'fileA'`" is not executed before `STOR` command).
B. *Code injection* An adversary injects malicious code into an existing file. This behavior is defined as "`STOR 'fileB'`" and executed after "`RETR 'fileB'`" in a session.
C. *Web accessibility confirmation* An adversary confirms that a created file is accessible on the web. This behavior is defined as an adversary uploading (`STOR`) a test file and trying to access it on the website (i.e., accessing "http://*ipaddr*/*filepath*," *ipaddr* is the IP address of our WCMS honeypot and *filepath* is the uploaded file's path assumed by an adversary), and the test file is immediately deleted (`DELE`) just after the access.
D. *Account validity confirmation* An adversary confirms that a stolen credential is valid. This behavior is defined as logout (e.g., "`QUIT`" or session terminating) without file access commands (e.g., `RETR`, `STOR`) after login.

Many cases of type A are for building stepping stones on a compromised server for other attack vectors, and this is introduced in Sects. 5.7.4 and 5.7.5. Our WCMS honeypot responds with an error reply to the HTTP request from outside hosts; therefore, all confirmation attempts of web content accessibility (type C) are failures. However, in almost all login sessions, adversaries continue malicious activity, such as code injection; therefore, we argue that this web content accessibility confirmation is just their reference information and does not affect adversary behavior.

The code injection event (type B) constitutes a large majority of sessions. A detailed injected code is discussed in Sect. 5.7.1. Distribution of the execution time of code injection is shown in Fig. 12. Much of the execution time of code injection is extremely short, for example, 78.4% of execution time of code injection[1] is shorter than 2 s. We also confirmed that command sequence patterns are not so diverse. Based on this evidence, many code injection events are automatically conducted using a certain toolkit.

[1] We used sessions of adversary group A, which was the most active in our monitoring, as shown in Fig. 11 and Table 1 and a large majority of our monitored events.

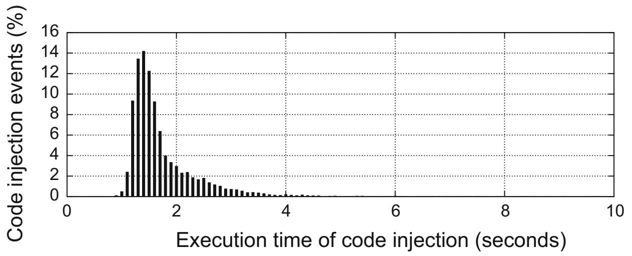

Fig. 12 Distribution of execution time of code injection. Execution time of code injection $T_{\text{injection}}$ is defined as $T_{\text{injection}} = T_{\text{stor}} - T_{\text{retr}}$, where T_{retr} is time of downloading file (`RETR` command), and T_{stor} is time of uploading (`STOR` command) compromised file, which was retrieved before

5.7 Malicious web content

We classify compromised web content and newly created web content obtained by our WCMS honeypot into the following categories: traffic redirection, server-side content, phishing page, and mass-mailing infrastructure. We assume that the compromised websites are used to achieve certain objectives of the adversary.

5.7.1 Injected code

Our WCMS honeypot can precisely extract injected code from compromised web content by using the `diff` command. Almost all injected codes are obfuscated JavaScript. This malicious obfuscated code is decoded on a web browser. It then automatically conducts redirection, e.g., creating an iframe tag. In addition, characteristic marker comments for an adversary, such as `<!-c3284d->` or `<!-6b1ee4->`, are added with obfuscated JavaScript. Adversaries are generally continuously developing obfuscation algorithms in order to circumvent detection; therefore, we must capture obfuscated codes and understand them in a timely manner. Our WCMS honeypot can continuously and automatically collect injected malicious codes. These codes must be actual good samples for supporting the generation of signatures or algorithms for detecting compromised web content.

5.7.2 Traffic redirection

Injected redirect codes lead to malicious websites prepared by adversaries; however, almost all of these redirect codes are obfuscated and unreadable. To disclose malicious websites behind the redirections, we used the web client honeypot to decode the obfuscation and access the next websites pointed to by the redirect codes. We confirmed that the web content of 305 out of 332 accounts had been injected with redirect codes to outside websites. In addition, the content was repeatedly injected with different redirect codes as time progressed. Most redirect destinations were malicious websites constructed using an *exploit kit*[2] or are hopping websites that redirect clients to them (Fig. 13). Both groups A and B used *traffic direction/distribution systems (TDSs)* as hopping sites. A TDS is used to direct traffic in order to sell pharmaceutical products, instigate search engine optimization (SEO) attacks, redirect users to adult sites, and redirect users to exploit websites for drive-by download-based malware infection [39,40]. Various exploit sites exist in the backend of TDSs. A TDS has filtering functionality based on client fingerprinting (e.g., browser, OS, IP geolocation, time frame, referral, local language settings) to block security inspections. The filtering functionality directs traffic unwanted by the adversary to popular websites. Adversaries use TDSs to conceal the final destination (i.e., exploit site). Injected redirect codes include only the URL of a TDS. By using a web client honeypot, HoneyCirculator can successfully obtain information on the TDS and also the final destinations.

However, there are some legitimate TDS providers, and not all TDS vendors are controlled by adversaries or sell their traffic to malicious entities. Two TDSs monitored in our experiment were obviously being used for drive-by downloads. We repeatedly conducted an additional inspection to extract these malicious websites. Because the TDS used by group A was composed of a *fast-flux service network (FFSN)*[3], we discovered a massive number of IP addresses that seemed to be bot-infected hosts. In contrast, the TDS used by group B had about 500 FQDNs, which changed as time progressed.

Many final destinations of redirection are malicious websites constructed using exploit kits. Exploit kits are known to have certain characteristics (e.g., file name, URL parameter, redirect chain), so we manually classified the exploit sites into the following five types: *Blackhole*, *Redkit*, *Phoenix*, *Incognito*, and *Neosploit*.

We confirmed that these adversary infrastructures used several exploit kits on their own websites. Although the redirect destinations designated by injected redirect code or TDS are regularly changed, HoneyCirculator can obtain information of newly malicious entities (e.g., obfuscated redirect code and FQDN/IP address of malicious website) when they are used. In this way, HoneyCirculator can immediately discover unknown malicious entities of specific adversary infrastructures without large-scale web crawling. However, we should be attentive when extracting URLs from compromised web content, as some redirect codes may be those

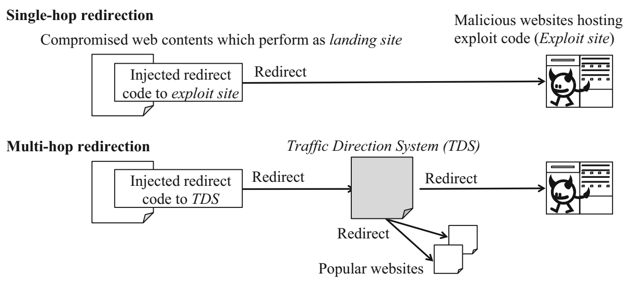
Single-hop redirection
Compromised web contents which perform as *landing site*
Malicious websites hosting exploit code (*Exploit site*)
Injected redirect code to *exploit site*
Redirect
Multi-hop redirection
Traffic Direction System (TDS)
Injected redirect code to *TDS*
Redirect
Redirect
Redirect
Popular websites

Fig. 13 Two types of monitored traffic redirections to malware distribution

for advertisements. That is, not all extracted URLs are those associated with drive-by downloads.

5.7.3 Web access control by server-side content

Some masqueraders put `.htaccess`, which is a configuration file for controlling web access, into their directories of the compromised web content. This `.htaccess` is used for traffic redirection. To circumvent crawling-based inspection, it checks the referrer of the accessed web client and permits redirection to malicious websites when the web client has a certain referrer. We confirmed that the URLs of a portal website or search engines are described in the referrer check routine. This means that only web clients from certain portal sites, search engine sites, or social networking service sites can be redirected to malicious websites. In other words, a web client honeypot directly accesses a compromised website, but it is not able to detect malicious websites. It also uses HTTP-error based redirection by using an *ErrorDocument* directive. If `.htaccess` has an " ErrorDocument 404 *redirect-URL*" directive, a user is redirected to the *redirect-URL* by an HTTP-302 redirect when he/she falsely accesses non-existing or error URLs. This technique does not require injecting a redirect code into original web content; therefore, it can circumvent being recognized by the legitimate WCMS administrator. When a web client mistakenly accesses a URL that is not found, it is redirected to an arbitrary URL, i.e., http://example.com/exploit.php in this case.

5.7.4 Phishing page

A newly created file is a phishing page that leaked credentials of *AOL*, *Gmail*, *Hotmail*, and *Yahoo* accounts. These pages contain an input form for user names and passwords. When a victim visiting this page inputs certain credentials and clicks the submit button, the credentials are sent to the adversary's e-mail address, which is embedded in the page. If we leak these actual services' credentials, newly compromised accounts on those services become a stepping stone of a new attack vector. This may result in our monitored attack cycle evolving to include new attack vectors, and monitoring

[2] A toolkit for constructing malicious websites that conduct drive-by downloads. Various types of exploit kits are traded in the underground economy [38].

[3] An FFSN uses both DNS round robin and short time-to-live (TTL) for a specific FQDN in order to have multiple IP addresses assigned to it. It is usually deployed by botnets.

of HoneyCirculator may be improved. Ensuring security on each actual service is beyond our control; therefore, a honeytoken-based monitoring system should cooperate with each actual service.

5.7.5 Mailing infrastructure

Another newly created file is e-mail-sending web content that is a CGI script and enables sending of spam/phishing e-mail. This CGI form allows an adversary to control all aspects of the message being sent: sender fields (e.g., *from*, *reply-to*, *sender name*), target addresses, and attachments. While we experimentally disclosed this web page for a certain period, the adversary who put the content accessed it and tried to send e-mail with a malicious hyperlink-URL leading to a malicious website constructed using the *Blackhole* exploit kit. This evidence means that e-mail is an additional way to distribute malware on our assumed attack cycle. However, our WCMS honeypot sinkholes all e-mail messages from itself because general public users would be exposed to mail-based threats if our WCMS honeypot permits mail sending. Security researchers should consider the legal and ethical aspects of security monitoring, as is the case in Sect. 5.7.4.

6 Evaluation

We evaluated our obtained data based on two aspects of effectiveness: (1) how many unknown malicious entities HoneyCirculator discovered and (2) how rapidly they are discovered.

6.1 Comparison with public blacklists

We compared the information we collected on adversaries and malicious websites with well-known public blacklists [23,41–45] (Table. 4). In this comparison, we manually selected FQDNs and IP addresses obviously recognized as exploit kits or TDSs from our collected information including accessed redirect destinations. We also selected masqueraders' IP addresses that had accessed stolen accounts on our WCMS honeypot. There were only ten overlapping IP addresses and no overlapping FQDNs. The results are listed in Tables 5 and 6. Most of our collected malicious FQDNs and IP addresses were not listed in other blacklists; the IP address overlap was only 4.5% (471/10,420), and the FQDN overlap was also 17.2% (155/900). These six blacklists must be widely used by security engineers and researchers. They have a certain level of quality; however, overlaps are quite small. These results indicate that HoneyCirculator can discover unique malicious entities that are not discovered by other blacklists.

We assume that the reasons the overlaps are so small are (1) the property of newly identified malicious entities and (2) the difference in the method of finding malicious entities. The former case arises from an FFSN and automatically generated domains. The TDS_A is composed of an FFSN mentioned in Sect. 5.7.2, which has about ten thousands of IP address. Generally, IP addresses of an FFSN are temporarily changed, many of which are bot-infected and are originally normal client hosts, so they tend not to register blacklists. Almost

Table 4 Public blacklists

Blacklist	# of IP addresses	# of FQDNs
MDL: MalwareDomainList	3498	3741
MP: MalwarePatrol	5457	6425
UBL: UrlBlackList (malware)	208, 801	111,945
MDB: MalwareDomainBlockList	3009	13,212
ZT: ZeuS Tracker	1672	1971
CMX: CleanMX (viruses)	65, 456	(n/a)

These IP addresses and FQDNs were registered in these blacklists from Mar. 2012 to Feb. 2013

Table 5 Comparison with public blacklists (IP address overlap)

Type of info.	Collected	∩ MDL	∩ MP	∩ UBL	∩ MDB	∩ ZT	∩ CMX
Masquerader	722	5	2	10	3	1	30
TDS_A	9476	2	11	55	1	2	136
TDS_B	33	7	0	10	3	0	6
Blackhole	24	15	1	3	5	0	12
Redkit	97	69	3	15	8	2	16
Phoenix	29	3	0	13	1	2	8
Incognito	18	7	1	1	1	1	0
Neosploit	19	7	0	5	1	2	8
Total	10, 420	113	18	102	21	8	209

Table 6 Comparison with public blacklists (FQDN overlap)

Type of info.	Collected	∩ MDL	∩ MP	∩ UBL	∩ MDB	∩ ZT	∩ CMX
Masquerader	(n/a)	(n/a)	(n/a)	(n/a)	(n/a)	(n/a)	(n/a)
TDS_A	84	0	0	31	5	0	(n/a)
TDS_B	525	3	0	19	11	0	(n/a)
Blackhole	127	3	0	0	0	0	(n/a)
Redkit	82	34	0	13	9	0	(n/a)
Phoenix	43	1	0	11	0	0	(n/a)
Incognito	32	2	0	5	5	0	(n/a)
Neosploit	7	1	0	11	0	0	(n/a)
Total	900	44	0	81	30	0	(n/a)

all domain names of TDS_B have both high linguistic randomness and static length (e.g., alphabetical 16 characters); therefore, they seem to be produced by a *domain generation algorithm (DGA)*. A DGA pseudo-randomly generates domain names based on a given particular timestamp, and a DGA-generated domain is used in a short period of time. Their domains also tend not to register blacklists based on above reasons. The latter cases arise from how conventional blacklisting methods collect information and what kinds of category blacklists register. Conventional methods collecting blacklist entities are known for web crawling, malware sandbox analysis, spamtrap, anti-malware telemetry, and so on. Discovered websites conducting drive-by download, C&C servers, and spam senders are listed in the blacklists. Conventional methods explore widely and shallowly over the Internet. In contrast, our system continuously tracks certain attack campaigns on the web-based attack cycle.

The small overlaps indicate we can enhance countermeasures by complementally combining the security knowledge obtained from our system and current blacklists.

6.2 Rapidity of malicious website discovery

We compared malicious websites discovered with our system and those of public blacklists. To evaluate the rapidity of discovery, we used *discovery latency* T_{latency}, which is defined as $T_{\text{latency}} = T_{\text{discover}} - T_{\text{register}}$, where T_{discover} is the time of domain discovery and T_{register} is the time of domain registration. A shorter discovery latency basically indicates rapid discovery of a malicious website. Each domain T_{discover} is described in the blacklists. The T_{register} is obtained from `whois` information. Note that it is possible that $T_{\text{latency}} < 0$ (T_{register} is newer than T_{discover}) because T_{register} is rewritable when the owner of a domain changes the domain information. The distributions of discovery latency are shown in Fig. 14 excluding those of domains that we could not obtain information of T_{register} from `WHOIS`. The results show that our discovered malicious websites' domains have considerably shorter discovery latency than those of blacklisted websites'

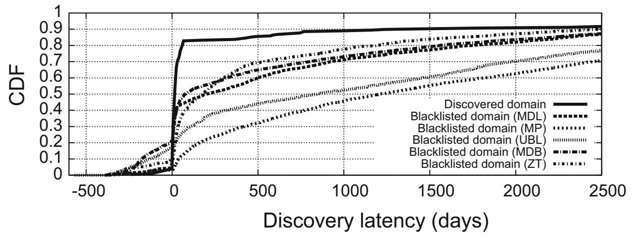

Fig. 14 Discovery latency of discovered malicious domain. We assumed why $T_{\text{latency}} < 0$ in some domains is that expired domain was registered again and *overwritten* registration time was later than discovery time

domains. Our proposed system can instantaneously discover malicious websites when they are used for an attack.

7 Discussion

7.1 Fidelity of decoy component

One of the main challenges with honeypots and malware sandbox involves camouflaging, in which they act as victim hosts. This is necessary so that the honeypot and malware sandbox can avoid being recognized by anti-analysis techniques used by adversaries. Adversaries often try to determine whether a target is an actual victim host or an analysis system from any unnatural victim behaviors.

We used IP address randomization and high-interaction systems in our experiment. To act as a victim host, web client honeypots use an actual OS and applications without any unnatural behavior such as an incomplete browser emulator, and our malware sandbox also runs on an actual OS and has Internet accessibility. Moreover, our network environment uses different autonomous systems (ASs) in order to randomize IP addresses. It also periodically changes the ASs; therefore, accesses of the web client honeypot and malware sandbox are from numerous IP addresses.

Another consideration in camouflaging is IP address consistency of the infected host. In the components of HONeyCirculator, the IP addresses of a host that collects malware (i.e., web client honeypot) and that of a host that leaks credentials (i.e., malware sandbox) are different. Therefore, an adversary can recognize them as security inspection systems by checking for IP address consistency. However, in recent years, the role of adversaries planning to spread malware infection has been subdivided into "distribute malware" and "control malware" (*Pay-Per-Install* [46]), and we, therefore, assume that a specific adversary does not always validate the consistency of IP addresses, or in other words, record all client IP addresses in every phase of an attack cycle (i.e., a host accessing a malicious website, a host downloading malware, and an infected host). In a dynamic IP address network environment, the user's IP address generally changes within a short period. Therefore, inconsistency between the IP address of an exploited host and that of an infected host is typical. Fortunately, because our experimental results also indicated that various adversary infrastructures accessed our monitoring system without suspicion, IP address consistency is not a serious problem.

If adversaries incur a higher cost, e.g., manual operation, it is possible to identify our components of HoneyCirculator, particularly our WCMS honeypot, as a decoy. For example, credentials from different victims involve a specific server that owns similar files on each account. Based on this commonality or similarity of a target server, cautious adversaries can discern and move away from our system, and this results in failure of our monitoring. This is the limitation on the practical side, although we can diversify the domain/IP address of a decoy server and original web content of our WCMS honeypot if we incur also higher cost. As mentioned in Sect. 5.6, code injection events seem to be automatically conducted using a certain toolkit. Adversaries behind masquerader hosts are compelled to automate the processes because they must simultaneously compromise a large amount of websites with stolen credentials over the Internet. Generally, manual operation to determine whether a compromised server is a decoy is time-consuming for adversaries. Fortunately, automated compromising based on an adversary's *dilemma* resulted in continuous and stable monitoring while our WCMS honeypot was not exposed as a decoy in most cases.

7.2 C&C over-blocking on malware sandbox

In our malware sandbox with semi-permeable Internet accessibility, only a few protocols for C&C communications are permitted, and other protocols are restricted for blocking malicious activity such as secondary malware infection. This type of *security-conscious* malware sandbox exhibits the C&C over-blocking problem because of restricted communications based on the policy. We should take into consideration the flexible control of C&C communication with safeguarding to improve monitoring. Fortunately, our experimental results indicated that various C&C communications worked successfully under our blocking policy. Although there is a possibility of falsely blocking C&C communications based on the original protocol, such as P2P, it is said that HTTP-based C&C communications are used by a large majority of malware families [47,48]. The C&C communication protocol must be based on a legitimate protocol; otherwise, intrusion detection systems will be able to easily detect it as protocol anomaly. We assume that the tendency of C&C using a legitimate protocol will continue in the near future. We believe that our malware sandbox is effective in exfiltrating bait credentials until the C&C communication protocols are changed.

7.3 Various methods for information exfiltration

In the proposed system, we preliminarily installed FTP client applications in the malware sandbox, and the malware sandbox set randomly generated credentials before analyzing the malware. Although malware automatically collects credentials and sends them to a remote host, the malware sandbox fails to actively leak credentials when the malware is triggered by a certain event. For example, a malware sandbox should launch a web browser to analyze malware that acts as a browser plug-in. Moreover, a malware sandbox should generate keystroke events when analyzing *keylogger*-type malware. A man-in-the-browser attack (MITB) can also be used to steal credentials. Through user interaction on the web browser, an MITB intercepts and manipulates transactions transparently between the web browser and online services. Our proposed system is limited to automatic information leakage without triggering the conditions of malware behavior such as complicated user interaction. BotSwindler [16] drives user interaction events based on a predefined scenario for actively leaking credentials.

7.4 Applicability to other applications/services

As mentioned in Sect. 3, the potential victims of information leakage are diverse. Our monitoring system can be applicable to various other applications/services that require ID/password authentication. A secure FTP (SFTP) only differs from FTP regarding encrypted communication; thus, it is the most straightforward application/service as a possible extension of the proposed monitoring system. In the same way, secure shell (SSH) is also a good candidate. We should change only two settings to apply them: preparing corresponding honeytokens (e.g., putting a configuration file of SFTP or SSH on a malware sandbox), and running the services on a WCMS honeypot awaiting masquerade attacks. However, to apply our monitoring system to public services (e.g., SNS, Online banking, Online shopping), we must cooperate with each service provider in order to monitor behind the service.

8 Conclusion

To achieve effective countermeasures to an attack cycle consisting of drive-by downloads, credential leakage, and compromised websites, we focused on tracking the usages of bait credentials leaked by malware and monitoring the activities of adversaries on a compromised web content management system. As an alternative to monitoring this kind of attack, we designed and implemented a monitoring system called HoneyCirculator that collects malware executables, actively leaks bait credentials, and lures adversaries to our WCMS honeypot. In a one-year experiment, our proposed system was successfully compromised by various adversary infrastructures without being recognized as a decoy, which allowed us to closely monitor adversary activities. The major advantage of our system is instantaneous discovery of unknown malicious entities even if they change redirection methods, malicious domains, exploit kits, and malware executables as long as they carry out the assumed attack cycle. In addition, the starting point of our monitoring was based on public blacklists; therefore, our system can detect new adversary infrastructures. Experimental results indicated that most of the information we collected was not contained in public blacklists; therefore, our proposed system was able to discover malicious activities in a different monitoring space from conventional blacklisting systems.

References

1. Websense Security Labs. Mass injection—nine-ball compromises more than 40,000 legitimate web sites. http://securitylabs.websense.com/content/Alerts/3421.aspx (2009)
2. Akiyama, M., Yagi, T., Aoki, K., Hariu, T., Kadobayashi, Y.: Active credential leakage for observing web-based attack cycle. In: Proceedings of the 16th International Symposium on Research in Attacks, Intrusions and Defenses (RAID2013), Springer
3. Moshchuk, A., Bragin, T., Gribble, S.D., Levy, H.M.: A crawler-based study of spyware on the web. In: 13th Annual Network and Distributed System Security Symposium (NDSS). ISOC (2006)
4. Provos, N., Mavrommatis, P., Rajab, M.A., Monrose, F.: All your iFRAMEs point to US. In: Proceedings of the 17th Conference on Security Symposium. USENIX (2008)
5. Stokes, J.W., Andersen, R., Seifert, C., Chellapilla, K.: WebCop: locating neighborhoods of malware on the web. In: Proceedings of the 3rd Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET'10). USENIX (2010)
6. The Honeynet Project: Know your enemy: malicious web servers. http://www.honeynet.org/papers/mws/ (2008)
7. Akiyama, M., Yagi, T., Itoh, M.: Searching structural neighborhood of malicious URLs to improve blacklisting. In: Proceedings of the 11th IEEE/IPSJ International Symposium on Application and the Internet (SAINT)
8. Invernizzi, L., Benvenuti, S., Cova, M., Comparetti, P.M., Kruegel, C., Vigna, G.: EvilSeed: a guided approach to finding malicious web pages. In: 2012 IEEE Symposium on Security and Privacy. IEEE (2012)
9. Zhang, J., Yang, C., Xu, Z., Gu, G.: Poison amplifier: a guided approach of discovering compromised websites through reversing search poisoning attacks. In: Proceedings of the 15th International Conference On Research In Attacks, Intrusions, And Defenses (RAID 2012). Springer (2012)
10. Spitzner, L.: Honeytokens: the other honeypot. http://www.symantec.com/connect/articles/honeytokens-other-honeypot (2003)
11. Bercovitch, M., Renford, M., Hasson, L., Shabtai, A., Rokach, L., Elovici, Y.: HoneyGen: an automated honeytokens generator. In: Proceedings of 2011 IEEE International Conference on Intelligence and Security Informatics (ISI). IEEE
12. Birk, D., Gajek, S., Gröbert, F., Sadeghi, A.-R.: Phishing phishers - observing and tracing organized cybercrime. In: Proceedings of the Second International Conference on Internet Monitoring and Protection (ICIMP). IARIA (2007)
13. Li, S., Schmitz, R.: A novel anti-phishing framework based on honeypots. Proceedings of the 2009 eCrime Researchers Summit (eCrime), IEEE, Tacoma, Washington, 20–21 Oct 2009
14. Yue, C., Wang, H.: BogusBiter: a transparent protection against phishing attacks. ACM Trans. Internet Technol. (TOIT) **10**(2), 31 (2010). [Article 6]
15. Bowen, B.M., Hershkop, S., Keromytis, A.D., Stolfo, S.J.: Baiting inside attackers using decoy documents. In: 5th International ICST Conference on Security and Privacy in Communication Networks (SecureComm2009). ICST (2009)
16. Bowen, B.M., Prabhu, P., Kemerlis, V.P., Sidiroglou, S., Keromytis, A.D., Stolfo, S.J.: BotSwindler: tamper resistant injection of believable decoys in VM-based hosts for crimeware detection. In: Proceedings of the 13th international conference on Research in Attacks, Intrusions, and Defenses (RAID 2010). Springer (2010)
17. Rist, L.: Know your tools: Glastopf. https://honeynet.org/files/KYT-Glastopf-Final_v1 (2010)
18. Canali, D., Balzarotti, D.: Behind the scenes of online attacks: an analysis of exploitation behaviors on the web. In: 20th Annual Network and Distributed System Security Symposium (NDSS). ISOC (2013)
19. Anubis (2014). http://analysis.seclab.tuwien.ac.at/
20. Malwr: https://malwr.com/
21. Shadow server : http://www.shadowserver.org/ (2014)
22. Kreibich, C., Weaver, N., Kanich, C., Cui, W., Paxon, V.: GQ: practical containment for measuring modern malware systems. In: Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference (IMC). ACM (2011)
23. Malware Domain List: http://malwaredomainlist.com/
24. Chenette, S.: Fireshark—A Tool to Link the Malicious Web. Blackhat Europe. Blackhat Europe, London (2010)
25. Akiyama, M., Aoki, K., Kawakoya, Y., Iwamura, M., Itoh, M.: Design and implementation of high interaction client honeypot for drive-by-download attacks. IEICE Trans. Commun. **E93–B**, 1131–1139 (2010)
26. Akiyama, M., Yagi, T., Kadobayashi, Y., Hariu, T., Yamaguchi, S.: Client honeypot multiplication with high performance and precise detection. IEICE Trans. Inform. Syst. **E98–D**, 775–787 (2015)

27. Seifert, C., Ramon, S.: Capture—honeypot client (Capture-HPC). https://projects.honeynet.org/capture-hpc (2008)
28. Wang, Y.-M., Beck, D., Jiang, X., Roussev, R., Verbowski, C., Chen, S., King, S.: Automated web patrol with strider honeymonkeys: finding web sites that exploit browser vulnerabilities. In: 13th Annual Network and Distributed System Security Symposium (NDSS). ISOC (2006)
29. Dell'Aera, A.: Thug: a new low-interaction honeyclient. (2012) http://www.honeynet.org/sites/default/files/files/HPAW2012-Thug
30. Nazario, J.: PhoneyC: a virtual client honeypot. In: Proceedings of the 3rd Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET'09). USENIX (2009)
31. Aoki, K., Yagi, T., Iwamura, M., Itoh, M.: Controlling malware HTTP communication in dynamic analysis system using search engine. In: Proceedings of the 3rd International Workshop on Cyberspace Safety and Security (CSS), (2011)
32. Chung, F.R.K.: Spectral graph theory. CBMS Regional Conference Series in Mathematics No. 92 (1997)
33. Chen, P.-Y., Hero, A.O.: Deep community detection. IEEE Trans. Sign. Process. **63**(21), 5706–5719 (2015)
34. Newman, M.: Spectral methods for network community detection and graph partitioning. Phys. Rev. E **88**, 042822 (2013)
35. Newman, M.: Networks: An Introduction. Oxford University Press Inc, New York (2010)
36. Spielman, D.A., Teng, S.-H.: Spectral partitioning works: planar graphs and finite element meshes. In: IEEE Symposium on Foundations of Computer Science. IEEE (1996)
37. Bureau, P.-M.: Same botnet, same guys, new code: Win32/Kelihos. (2011)
38. Grier, C., Ballard, L., Caballero, J., Chachra, N., Dietrich, C.J., Levchenko, K., Mavrommatis, P., McCoy, D., Nappa, A., Pitsillidis, A., Provos, N., Rafique, M.Z., Rajab, M.A., Rossow, C., Thomas, K., Paxson, V., Savage, S., Voelker, G.M.: Manufacturing compromise: the emergence of exploit-as-a-service. In: Proceedings of the 19th ACM Conference on Computer and Communication Security. ACM (2012)
39. Symantec. Web-based malware distribution channels: a look at traffic redistribution systems. http://www.symantec.com/connect/blogs/web-based-malware-distribution-channels-look-traffic-redistribution-systems (2011)
40. Traffic direction systems as malware distribution tools: http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/reports/rpt_malware-distribution-tools (2011)
41. Clean, M.X.: http://support.clean-mx.de/clean-mx/viruses (2014)
42. DNS-BH 2014. Malware domain blocklist. http://www.malwaredomains.com/
43. Malware Patrol : http://www.malware.com.br/
44. URLBlackList. http://urlblacklist.com/ (2014)
45. ZeuS Tracker. https://zeustracker.abuse.ch/ (2014)
46. Caballero, J., Grier, C., Kreibich, C., Paxson, V.: Measuring pay-per-install: the commoditization of malware distribution. In: Proceedings of the 20th USENIX Security Symposium. USENIX (2011)
47. Nappa, A., Rafique, M.Z., Caballero, J.: FIRMA: malware clustering and network signature generation with mixed network behaviors. In: Proceedings of the 16th International Symposium on Research in Attacks, Intrusions and Defenses (RAID2013). Springer (2013)
48. Nelms, T., Perdisci, R., Ahamad, M.: ExecScent: mining for new C&C domains in live networks with adaptive control protocol templates. In: Proceedings of the 22nd USENIX Conference on Security. USENIX (2013)