The Art of False Alarms in the Game of Deception: Leveraging Fake Honeypots for Enhanced Security

Apostolis Zarras
Ruhr-University Bochum
apostolis.zarras@rub.de

***Abstract*—The great popularity of the Internet increases the concern for the safety of its users as many malicious Web pages pop up in daily basis. Client honeypots are tools, which are able to detect malicious Web pages, which aim to infect their visitors. These tools are widely used by researchers and anti-virus companies in their attempt to protect Internet users from being infected. Unfortunately, cyber-criminals are becoming aware of this type of detection and create evasion techniques that allow them to behave in a benign way when they feel to be threatened. This bi-faceted behavior enables them to operate for a longer period, which translates in more profit. Hence, these deceptive Web pages pose a significant challenge to existing client honeypot approaches, making them incapable to detect them when exhibit the aforementioned behavior.**

In this paper, we mitigate this problem by designing and developing a framework that benefits from this bi-faceted behavior. Our main goal is to protect users from being infected. More precisely, we leverage the evasion techniques used by cyber-criminals and implement a prototype, called SCARECROW, which triggers false alarms in the cases of deceptive Web pages. Consequently, the users that use SCARECROW for Web surfing can remain protected, even if they visit a malicious Website. We evaluate our implementation against malicious URLs provided by a large anti-virus company and show that when SCARECROW is deployed, malicious Websites with bi-faceted behavior do not launch their attacks against normal users.

Keywords*—*Client Honeypots; Malicious Websites; Deception

I. INTRODUCTION

Over the last decade, the Internet has become extremely popular. In daily basis, individuals utilize the Internet for a variety of tasks, ranging from education and work to personal entertainment and social networking. In this Internet-connected society, users spend most of their online time using browsers to access the Internet. This tendency makes the browsers the most indispensable software product of our days. Unfortunately, this enormous growth of Internet's popularity has drawn the attention of miscreants that try to illegally monetize their activities. For this purpose, cyber-criminals create fraudulent Websites that lure Internet users and force them to install malicious software (malware) on their computers, most of the times without the users' prior knowledge or consent. These Websites usually exploit vulnerabilities in a browser and trick it to install malware on the user's machine [12, 27, 28]. The infected computers, known as bots, are typically organized into so-called botnets that are remotely controlled by a single entity, known as the botmaster [1, 6, 7]. This is one of the most profitable businesses in the underground market in which botmasters can gain a significant amount of revenue per year by operating only few thousand bots [15, 33].

As a primary line of defense against this emerging threat, researchers and security analysts utilize client honeypots to discover, study, and obliterate malicious Websites. Client honeypots—in contrast to server honeypots, which are decoy systems set up to attract and trap attackers that attempt to penetrate them—crawl the Internet, interact with servers, and classify Websites based on their malicious behavior. More precisely, client honeypots, which are usually instrumented virtual machines, visit a Web page and monitor all the changes in a virtual machine's file system, configuration settings and running processes. If they notice an unexpected behavior, the Web page is flagged as malicious [23, 24, 37]. The secure environment where the client honeypots operate, allow the researchers to discover attacks and exploits, release security patches, and create browser's alerts that triggered when a user tries to access a malicious Website. The findings are usually published on blacklists, which in turn are used for further study or development of more secure Web products. For instance, modern browsers utilize mechanisms such as Google's Safe Browsing [11] for protecting their users against known Web threats and offer them a more secure Web surfing experience.

Client honeypots pose a significant challenge for the smooth operation of fraudulent Websites. Unfortunately, cyber-criminals who watched the incoming traffic of their Websites to exponentially shrink developed techniques to avoid detection. More precisely, to overcome detection, many malicious Web pages established a bi-faceted behavior. By adopting a series of inspections, deceptive Websites can accurately determine if a client is a normal browser or a client honeypot. In detail, when a client is probed, it returns back a response. This response is valuable for the attackers as they can gain information about the true origin of the client. Thus, to avoid detection, they do not mount their attack if they are dealing with a client honeypot but instead they appear to have a completely benign behavior. This strategy allows them to remain hidden and operate for a longer period of time.

Several of the evasion techniques used by malicious Web pages have been previously studied [16]. In general, cyber-criminals introduce a number of attacks that leverage the weaknesses in the design of client honeypots. These attacks are divided in two major categories: (i) identification of the monitoring environment, and (ii) evasion of its detection. The malicious Web pages in order to remain undetected must successfully implement attacks from both categories. While the attacks from the first category allow the miscreants to detect the presence of a monitoring system, the attacks from the second category transform the malicious behavior of the Web pages to appear as completely benign.

In this paper we utilize the bi-faceted behavior of malicious Websites to enhance the users' security. We designed and implemented a framework, called SCARECROW, which benefits from the precautions taken by deceptive Websites to hide their malicious activities from client honeypots. In particular, SCARECROW cloaks a Web browser to appear as a monitoring system. More precisely, when a user visits a Website, the framework employs all the necessary mechanisms to disguise its actions as they are generated by a client honeypot. Thereby, all interactions between a user's browser and a Website take place under this security umbrella. Unfortunately, this may occasionally interfere with the normal browsing experience. Nevertheless, users can easily deactivate the framework on demand, or can configure it to meet their needs. In summary, SCARECROW allows the users to surf the Internet protected from attacks that exploit their browsers and download malware, transforming eventually their computers into bots.

We implemented our prototype as an extension for Firefox browser. Browser extensions offer a series of advantages as they require minimum effort from users and are already widespread, as most of the Internet users have at least one or more extensions installed in their browsers. Having that in mind, we implemented SCARECROW as a framework with a clear attack *prevention* focus, which is designed to be easily integrated with Firefox and thus, can be used by inexperienced users. This contrasts with common client honeypot approaches, used by researchers and security experts, which have a strict attack *detection* mission. Finally, we evaluated our implementation against malicious URLs provided by a large anti-virus company and show that our framework can protect users against malicious Websites that display a bi-faceted behavior. We should note that our framework does not replace traditional anti-virus products and techniques, as there is a wide-variety of Web pages that have exactly the same behavior for both normal browsers and client honeypots. Nevertheless, SCARECROW constitutes the first line of defense against infections from *intelligent* malicious Web pages that try to remain undercover.

In summary, we make the following main contributions:

- We transform a traditional attack detection approach, such as client honeypots, to a system with a clear attack prevention focus.
- We propose SCARECROW, a novel framework, which enhance users' security when surfing the Web. This framework allows the users to surf the Web protected from attacks that performed by deceptive Web pages.
- We implement a prototype of our approach as a browser extension for the Firefox browser. Our prototype creates events that trigger the inspection mechanisms against client honeypots used by intelligent adversaries.
- We evaluate our implementation and our preliminary results show that SCARECROW can successfully protect users against malicious Websites that display a bi-faceted behavior.

II. CLIENT HONEYPOTS

Client honeypots, also known as honeyclients, constitute a security technology capable of discovering malicious servers on Internet. More specifically, the client honeypots pose as normal Web clients and interact with servers to investigate whether an attack has occurred. They divided in: (i) low-interaction, (ii) high-interaction, and (iii) hybrid client honeypots. In the rest of this section we briefly describe these categories.

A. Low-Interaction Client Honeypots

Low-interaction client honeypots use simulated clients, similar to Web crawlers, whose purpose is to interact with a server. Their task is to analyze the server's responses and determine about its nature. By deploying static analysis techniques, such as signatures, can search for malicious patterns and assess whether an attack has occurred.

Increased speed and low resource consumption are the major advantages of the low-interaction client honeypots. However, since they are usually signature-based approaches, they are unable to detect previously unseen attacks such as zero-day threats. Additionally, their simplicity makes them easily distinguishable by advanced exploits.

B. High-Interaction Client Honeypots

High-interaction client honeypots are fully functional systems comparable to real systems. They use a full-featured Web browser to visit potentially malicious Web pages and classify their behavior. To this end, they monitor the environment in which the browser operates to inspect any modification of the system's state after visiting a Web server. The detection of any change in the monitored environment indicates the occurrence of an attack, and the corresponding Web page is flagged by the system as malicious.

This type of client honeypots is very effective at detecting novel attacks against clients. Nevertheless, the tradeoff for this accuracy is the high complexity of running a high-interaction honeypot and the time consuming monitoring process. Additionally, since the client honeypots are running inside virtual machines, the malicious Web pages may try to detect the presence of the virtual environment and cease from launching the attack. Consequently, because of the fact that no detectable state change in the monitored environment occurred, the honeypot is likely to incorrectly classify the server.

C. Hybrid Client Honeypots

Low-interaction and high-interaction client honeypots try to detect malicious Web pages from a different perspective. Hence, the combination of both approaches leads to a detection system that integrates high speed and the ability to identify new threats. This detection system is called hybrid client honeypot.

Hybrid client honeypots combine the advantages of both low-interaction and high-interaction client honeypots. More precisely, they incorporate the classification methods used by low-interaction and high-interaction client honeypots into a hybrid system, which is capable of identifying malicious Web pages in a cost effective way on a large scale. For that reason, the hybrid client honeypot approach outperforms a high-interaction client honeypot with identical resources and identical false positive rate.

III. System Overview

In this section, we discuss about the architecture of our proposed system. We initially explain the threat model we use throughout this paper and then give information on the design details of SCARECROW.

A. Threat Model

We assume that a user surfs the Internet with a vulnerable Web browser and visits a malicious Web page. Even if the browser itself is secure, there exist a variety of extensions installed in the browser that might have subtle vulnerabilities, which can be exploited. In fact, hundreds of these extensions' vulnerabilities have been studied in prior works [3,4], and tools that automatically highlight these weaknesses have already been implemented [2, 17, 20, 26]. Unfortunately, this is not a hypothetical scenario but an everyday situation that millions of users face every day while surfing the Web.

Additionally, we assume that the malicious Web page is aware of the browser's, or extensions', vulnerabilities and possesses all the necessary tools, which can exploit the browser. We believe that the purpose of this page is to generate profit for its operator and thus, the more time remains undetected the more profit it creates. Consequently, we consider that the page is capable of displaying only benign content when identifies the presence of a client honeypot.

B. Design Details

Since adversaries' goal is to hide their malicious activities from automated detection systems, they created mechanisms to successfully inform them when client honeypots visit their Web pages. Kapravelos et al. [16] created a list of these mechanisms, which is divided in two categories: (i) identification of the monitoring environment, and (ii) evasion of its detection. Table I provides an overview of the techniques used in each category. SCARECROW leverages the heuristics of both categories to misinform miscreants for the true identity of a client. In the following, we provide details about the integration of each mechanism in our prototype.

Virtual Machine Detection. Since many successful drive-by-download attacks install malware that interferes with the victim's operating system, many client honeypots utilize virtual machines to protect the actual host from being infected. As a matter of fact, after a complete scan to an *under investigation* Web page has been performed, the virtual machine returns the operating system to a safe state. Unfortunately, the attackers in order to detect a monitor environment utilize mechanisms, which can reveal the presence of a virtual machine.

In our system, we turn this knowledge into a protection heuristic. More precisely, some virtual machines have the tendency to reveal their presence by inserting elements into the guest operating system, which eventually can be traced. For instance, these elements can be service processes, unique files or directories, or even specific registry keys. SCARECROW is able to generate all the required files in the operating system, as well as, dummy executable files, which then execute. This way, it creates processes that cloak the operating system to appear as it is running inside a virtualized environment.

Table I: Popular mechanisms used by malicious Web pages to evade detection.

Categories of Malicious Mechanisms	Heuristics
Monitoring Environment Detection	Virtual Machine Detection Client Honeypot Detection HTTP Headers Checks
Detection Evasion	Mouse Events Exploitation Whitelist Manipulation

Client Honeypot Detection. An adversary in order to detect the presence of a monitoring system can actually check for signs of the client honeypot itself. These signs can be, for example, executable or DLL files. Sadly, there exist honeypots that do not try to hide their presence, and even worse the attackers are familiar of this practice. In theory, a malicious Web page can use the JavaScript engine to load a *suspicious* file from the client's local file system, which is only appeared in cases of client honeypots. This file can be an executable or library, since the engine does not perform any checks to validate the type of the files that have been requested. If the file exists, the attacker is getting aware of the situation.

Hopefully, Same-Origin Policy [36] prohibits the access on local files through JavaScript. Nevertheless, there exist some older browsers' versions, or some customized browsers used by client honeypots that allow this technique to be executed. Similar, SCARECROW has the ability to hook on specific JavaScript events that request access to the filesystem. In detail, each time a Web page tries to access a local file, a warning informs the user about the intention of the script. Additionally, SCARECROW creates several client honeypots' executable files on every browser's startup and deletes them on every browser's shutdown. We observed that only the presence of these files is sufficient to trigger an alert.

HTTP Headers Checks. Whenever a browser visits a Web page, it sends one or multiple HTTP requests to the equivalent Web server. Each request contains fields, called headers, which reveal information about the type of request, the connection, the browser etc. Among these fields, there are two specific headers that prove to be quite useful for the attackers: the *User-Agent* and the *Referer*. While the *User-Agent* can reveal information about the browser itself, the *Referer* provides the source URI from which the call of the current Web page originated. An adversary could utilize this knowledge to determine if the client is actually a honeypot. Modern honeypots can easily modify the *User-Agent* value to mimic a normal Web browser, however, it is difficult to predict the correct origin of the hosted URI, based on which the malicious Web page will trigger the attack.

SCARECROW modifies both HTTP headers to appear as a misconfigured client honeypot. For this purpose, since there is only a small portion of malware that targets non-windows operating system, we use *User-Agent* header to classify the browser as a client honeypot, which runs inside a Linux-based machine. Additionally, we adjust the *Referer* header to purport as the Web page is harvested from an anti-virus vendor. Consequently, as the user is probably visiting the malicious Web page by clicking a link in a different Website, this information will not be revealed to the attacker.

Mouse Events Exploitation. To ensure that they deal with a real client, many malicious Web pages wait for a prior user input before launching an attack. Since frequently a client honeypot only visits a Web page, without actively interact with it, events such as the movement of the mouse will never be fired. Therefore, adversaries can hold their attacks before one or more similar events convince them that an actual user generates the traffic to their Web server and not an automated tool as a Web crawler or a client honeypot. For this purpose they utilize JavaScript. In particular, some JavaScript objects have events associated with them. Usually, these events are user actions, or at least initiated by a user, for instance, click on an object or simply moving the mouse. Hence, an *event listener* can be used in JavaScript code to specify actions in response to the occurrence of a specific event.

SCARECROW prevents the *event listeners* from catching the mouse movements. Consequently, the adversaries get the impression of dealing with client honeypots instead of real users. However, we believe that this might affect the user experience, especially in cases where the mouse interaction is necessary. An example could be some online flash games that require the users' mouse events to operate.

Whitelist Manipulation. All browsers are able to interact with the operating system. This is an important action for procedures that store browsing data on the hard disk drive or load additional programs to display Web content. Client honeypots use the so-called *whitelists* when analyzing attacks to separate the harmless interactions between the browser and the operating system, from the harmful. Adversaries circumvent these *whitelists* with cache poisoning attacks. Since creating, reading and writing to a file in the browser cache is a whitelist action, an attacker could leverage it to change the files in the cache, in order to force the redirection from benign URLs to malicious Web pages. This attack is so severe that even if a victim closes or reboots the browser, it is sufficient just a visit to any Web page that loads a modified cached script to re-infect the machine.

To protect a client from these redirections we clear the cache on a regular basis. The browser's cache is a data space where Web pages are stored once they are loaded. If a Web page gets revisited in the future, the content of this page can be loaded directly from the cache, which is faster than loading the content from a remote server. Each Web page is accompanied with the duration on which it can be loaded from the cache, before it needs to be downloaded again from the server. An adversary could inject malicious code inside a Web page and change its expiration date to the far future, so that is always being loaded directly from the cache. When we clear the cache regularly we cannot prevent the original injection of malicious code, however, we can prevent a re-infection from happening.

To sum up, we consider the combination of the aforementioned mechanisms sufficient to delude a bi-faceted malicious Web page so not to launch its attack. However, one can notice that mixed information is returned from the different components of SCARECROW. While some components claim, for instance, that the operating system is a Linux distribution, others state that all the processes run inside a Windows OS. Note that this is not a wrong implementation of SCARECROW, but is designed in this way intentionally, so to appear to attackers as a misconfigured client honeypot.

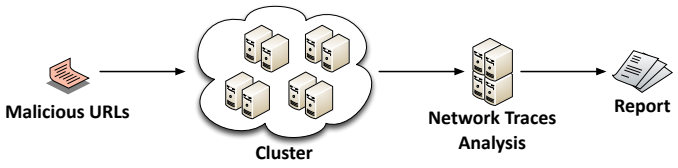

Figure 1: Overview of the experimental environment.

IV. EXPERIMENTAL EVALUATION

In this section we present the evaluation results of our prototype. First, we briefly describe the experimental environment and then evaluate the protection effectiveness of SCARECROW using real malware samples from a large anti-virus company.

A. Experimental Environment

We performed our experiments on a cluster that consisted of Windows XP machines. We chose Windows XP as the hosts' operating system due to its known vulnerabilities, which make it a perfect target for adversaries. Furthermore, it is long been known that Java and Flash are favored targets of attackers thanks to their huge installation bases and numerous security issues. Thus, in order to increase the chance of successful attacks against our infrastructure, in each host we installed some older versions of Java, Flash, Acrobat, and VLC that are susceptible to security breaches. Since we implemented the current version of our prototype as a Firefox extension, we decided to run all our experiments with this Web browser in order to have a consistency in our results. To this end, we installed SCARECROW only on half of the machines of our cluster, while to the rest we installed a vanilla version of Firefox.

For our experiments, we used a dataset of $8,291$ malicious URLs, which are provided by a large anti-virus company. We visited each URL with two machines from the cluster, one with SCARECROW installed on it and one without, generating in parallel events such as mouse movements and keystrokes. In addition, we captured and stored the network traffic of each visit for further analysis. After each Web browser's visit to the malicious URL, we returned the host to a clean state. To do so, we used *Clonezilla* [5] as a disk recovery solution. Albeit that using a disk recovery solution is a time consuming task, compared to virtual machines snapshots, we rejected the snapshots because they could have interfered with the *Virtual Machine Detection* heuristic of SCARECROW causing inaccurately results on our experiments. After visiting each URL, we analyzed the captured network traffic looking for existence of malicious traces. In case that the traffic of both machines contained malicious traces we concluded that either the malicious Web page did not display a bi-faceted behavior, or our system was not able to deceive attackers' mechanisms. Otherwise, if the captured traffic of the vanilla browser contained malicious traces, but the traffic from SCARECROW was clean, we accounted this case as a successful protection.

In summary, Figure 1 displays the overview of our experimental environment. Web browsers visit the malicious URLs, while the captured traffic is forwarded for extensive analysis that assesses the effectiveness of SCARECROW.

B. Protection Effectiveness

We evaluated the protection effectiveness of SCARECROW against real malicious Web pages. From the $8,291$ malicious URLs that constitute our dataset, SCARECROW successfully prevented the infection from 437. The outcomes of this experiment are two-fold. On the first hand, the results reveal that there is a portion of malicious Web pages in the Internet that try to avoid detection by appearing a benign behavior when they are visited by a client honeypot. On the other hand, they show that is possible to use attackers' precautions for users' benefit by simply camouflaging a normal web browser to a client honeypot.

We then analyzed, the remaining $7,854$ malicious URLs that SCARECROW could not prevent them from infecting the host machines. We wanted to see if these infections caused because SCARECROW was not able to prevent them, or because the malicious Web pages did not try to hide their existence from client honeypots. Thus, we carefully examined their source code searching for indications of possible bi-faceted behavior. Nevertheless, we could not find signs that reveal any attempt of concealing their malicious essence. This is an interesting result, which shows that the majority of miscreants are not care of hiding their fraudulent activities. We can only speculate that most of the operators of these Web pages do not have advanced technical knowledge and thus, in order to launch their attacks use black-market tools, which may not support protection against client honeypots.

V. LIMITATIONS

Despite the fact that SCARECROW is able to effectively prevent attacks from deceptive Web pages, like any other system has its own limitation. An adversary, who gains knowledge on our system existence, might be able to effectively infect the users. In the following, we discuss our system's limitations.

A. User Interaction Interference

The most obvious limitation of SCARECROW is the problems that might cause in the users' browsing experience in specific Web pages. Although, usually a user does not notice any difference while surfing the Internet, there are some cases in which our system interfere with the user's actions. One particular case is when the user tries to play an online flash game that requires input from the user's mouse movements. Albeit the user can see the mouse cursor moving in the screen, SCARECROW prevents these mouse events from triggering the corresponding *event listeners*. Consequently, the result of this restriction is the mouse movement events never reach the game engine, which assumes that the user has never provided any valid input.

A possible solution on this problem would be the deactivation of the mouse events restrictions. As a matter of fact, SCARECROW allow to its users to select the heuristics they want to enable. Additionally, they have the capability to enable/disable individual heuristics for distinct Websites. This way, the restriction of mouse events can be disabled for a Website that requires this specific user input, and be enabled for the rest of the Websites. However, the users should be certain that the visited Web page does not contain malware before deactivating any of the heuristics.

B. File Content Verification

One of SCARECROW's heuristics is the creation of dummy executable and DLL files that used by virtual machines and client honeypots. Most of the attackers only check for the existence of these files. However, an intelligent deceptive Web page can validate the provided with the expected files. If there is no match, it can assume that is under a deception attempt. In that case, having that knowledge, it can decide whether it proceeds with the attack or not.

In this case, we can easily overcome this pitfall by simply providing the real files. Nevertheless, as we did not want to force users to save additional executable files and libraries to their computers, we offer them the opportunity to select between the dummy and the real files when using our system. If they choose the second option, SCARECROW downloads all the required files from an online repository.

VI. RELATED WORK

Several malware detection systems focus on network flow analysis [8, 13] or require deep packet inspection [14] in order to detect compromised machines within a local network. Other detection approaches aim to identify common behaviors of infected machines when performing malicious activities [10, 13, 34, 38]. On the other hand, traditional honeypots as a detection approach have proven very successful with tasks as identifying malware [21], creating intrusion detection signatures [19] and understanding distributed denial-of-service (DDoS) attacks [22]. As successful successors, client honeypots visit Websites and monitor changes in the underlying operating system that may be caused by malicious Web pages [23, 24, 27, 35, 37].

Attacks against client honeypots is an aged old idea. Wang et al. [37] mentioned possible evasions techniques against HoneyMonkey. Rajab et al. [29] in their study revealed that client honeypots, among other detection system, could not protect themselves against evasive techniques deployed by attackers. Kapravelos et al. [16] examined the security model that high-interaction client honeypots utilize, and evaluated their weaknesses against intelligent evasion techniques.

The concept of deceiving malware to hide its malicious behavior and thus, not to harm a machine is not new. Researchers suggested the use of fake honeypots to scare the attackers [32]. Additionally, Rowe [31] presented a study in which he assesses several tools for evaluating honeypot deceptions. Finally, Garg and Grosu [9] proposed a game theoretic framework for modeling deception in honeynets.

Researchers utilize browser extensions when their deployed systems focus on inexperienced users with a clear intention of increasing users' security and privacy while surfing the Internet. PwdHash [30] is a browser extension designed to improve password authentication on the Web with minimal change to the user experience and no change to existing server configurations. Papadopoulos et al. [25] presented an obfuscation-based approach that enables users to follow privacy-sensitive channels on microblogging services, while, at the same time, making it difficult for the services to discover users' actual interests. Kontaxis et al. [18] proposed a design for privacy-preserving social plugins that decouples the retrieval of user-specific content from the loading of a social plugin.

VII. Conclusions

Malicious Web pages remain the primary mean used by miscreants to spread malware. Unfortunately, the effectiveness of existing detection approaches, such as client honeypots, to correctly classify malicious Web pages are considered modest, especially in cases where the attackers deployed techniques to display a benign behavior against these detection mechanisms. In this paper, we benefit from the attackers' precautions and we transform these detection approaches to a system with a clear attack prevention focus. In particular, we designed and implemented a prototype that triggers false alarms causing deceptive Web pages to display a benign behavior. Our primary results indicate that the users, which utilize our system, can be protected from these attacks without any additional anti-virus software installed on their machines.

References

[1] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2006.

[2] S. Bandhakavi, S. T. King, P. Madhusudan, and M. Winslett. VEX: Vetting Browser Extensions for Security Vulnerabilities. In *USENIX Security Symposium*, 2010.

[3] A. Barth, A. P. Felt, P. Saxena, and A. Boodman. Protecting Browsers from Extension Vulnerabilities. In *ISOC Network and Distributed System Security Symposium (NDSS)*, 2010.

[4] N. Carlini, A. P. Felt, and D. Wagner. An Evaluation of the Google Chrome Extension Security Architecture. In *USENIX Security Symposium*, 2012.

[5] Clonezilla. The Free and Open Source Software for Disk Imaging and Cloning. http://clonezilla.org, Jul 2014.

[6] E. Cooke, F. Jahanian, and D. McPherson. The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets. In *USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, 2005.

[7] D. Dagon, G. Gu, C. P. Lee, and W. Lee. A Taxonomy of Botnet Structures. In *Annual Computer Security Applications Conference (ACSAC)*, 2007.

[8] J. François, S. Wang, T. Engel, et al. BotTrack: Tracking Botnets Using NetFlow and PageRank. In *IFIP Networking Conference*, 2011.

[9] N. Garg and D. Grosu. Deception in Honeynets: A Game-Theoretic Analysis. In *Annual IEEE SMC Information Assurance and Security Workshop (IAW)*, 2007.

[10] J. Goebel and T. Holz. Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation. In *USENIX Workshop on Hot Topics in Understanding Botnet (HotBots)*, 2007.

[11] Google. Safe Browsing API. https://developers.google.com/safe-browsing, Jul 2014.

[12] C. Grier, L. Ballard, J. Caballero, N. Chachra, C. J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis, et al. Manufacturing Compromise: The Emergence of Exploit-as-a-Service. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

[13] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection. In *USENIX Security Symposium*, 2008.

[14] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *USENIX Security Symposium*, 2007.

[15] C. Kanich, N. Weaver, D. McCoy, T. Halvorson, C. Kreibich, K. Levchenko, V. Paxson, G. M. Voelker, and S. Savage. Show Me the Money: Characterizing Spam-Advertised Revenue. In *USENIX Security Symposium*, 2011.

[16] A. Kapravelos, M. Cova, C. Kruegel, and G. Vigna. Escape from Monkey Island: Evading High-Interaction Honeyclients. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. 2011.

[17] R. Karim, M. Dhawan, V. Ganapathy, and C.-c. Shan. An Analysis of the Mozilla Jetpack Extension Framework. In *European Conference on Object-Oriented Programming (ECOOP)*, 2012.

[18] G. Kontaxis, M. Polychronakis, A. D. Keromytis, and E. P. Markatos. Privacy-Preserving Social Plugins. In *USENIX Security Symposium*, 2012.

[19] C. Kreibich and J. Crowcroft. Honeycomb: Creating Intrusion Detection Signatures Using Honeypots. *ACM SIGCOMM Computer Communication Review*, 34(1):51–56, 2004.

[20] B. S. Lerner, L. Elberty, N. Poole, and S. Krishnamurthi. Verifying Web Browser Extensions' Compliance with Private-Browsing Mode. In *European Symposium on Research in Computer Security (ESORICS)*, 2013.

[21] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer Worm. *IEEE Security & Privacy*, 1(4):33–39, 2003.

[22] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage. Inferring internet denial-of-service activity. *ACM Transactions on Computer Systems (TOCS)*, 24(2):115–139, 2006.

[23] A. Moshchuk, T. Bragin, D. Deville, S. D. Gribble, and H. M. Levy. SpyProxy: Execution-based Detection of Malicious Web Content. In *USENIX Security Symposium*, 2007.

[24] A. Moshchuk, T. Bragin, S. D. Gribble, and H. M. Levy. A Crawler-based Study of Spyware in the Web. In *ISOC Network and Distributed System Security Symposium (NDSS)*, 2006.

[25] P. Papadopoulos, A. Papadogiannakis, M. Polychronakis, A. Zarras, T. Holz, and E. P. Markatos. k-subscription: Privacy-Preserving Microblogging Browsing Through Obfuscation. In *Annual Computer Security Applications Conference (ACSAC)*, 2013.

[26] J. G. Politz, S. A. Eliopoulos, A. Guha, and S. Krishnamurthi. ADsafety: Type-Based Verification of JavaScript Sandboxing. In *USENIX Security Symposium*, 2011.

[27] N. Provos, P. Mavrommatis, M. Abu Rajab, and F. Monrose. All Your iFRAMEs Point to Us. In *USENIX Security Symposium*, 2008.

[28] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, N. Modadugu, et al. The Ghost in the Browser: Analysis of Web-based Malware. In *USENIX Workshop on Hot Topics in Understanding Botnet (HotBots)*, 2007.

[29] M. Rajab, L. Ballard, N. Jagpal, P. Mavrommatis, D. Nojiri, N. Provos, and L. Schmidt. Trends in circumventing web-malware detection. *Google, Google Technical Report*, 2011.

[30] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger Password Authentication Using Browser Extensions. In *USENIX Security Symposium*, 2005.

[31] N. C. Rowe. Measuring the effectiveness of honeypot counter-counterdeception. In *System Sciences, 2006. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on*, 2006.

[32] N. C. Rowe, E. J. Custy, and B. T. Duong. Defending Cyberspace with Fake Honeypots. *Journal of Computers*, 2(2):25–36, 2007.

[33] B. Stone-Gross, R. Stevens, A. Zarras, R. Kemmerer, C. Kruegel, and G. Vigna. Understanding Fraudulent Activities in Online Ad Exchanges. In *ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2011.

[34] G. Stringhini, M. Egele, A. Zarras, T. Holz, C. Kruegel, and G. Vigna. B@bel: Leveraging Email Delivery for Spam Mitigation. In *USENIX Security Symposium*, 2012.

[35] The Honeynet Project. Capture-HPC: Client Honeypot / Honeyclient. https://projects.honeynet.org/capture-hpc, Jul 2014.

[36] W3C. Same-Origin Policy. http://www.w3.org/Security/wiki/Same_Origin_Policy, Jul 2014.

[37] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King. Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities. In *ISOC Network and Distributed System Security Symposium (NDSS)*, 2006.

[38] A. Zarras, A. Papadogiannakis, R. Gawlik, and T. Holz. Automated Generation of Models for Fast and Precise Detection of HTTP-Based Malware. In *Annual Conference on Privacy, Security and Trust (PST)*, 2014.