Analyzing and Defending Against Web-based Malware

JIAN CHANG, University of Pennsylvania
KRISHNA K. VENKATASUBRAMANIAN, University of Pennsylvania
ANDREW G. WEST, University of Pennsylvania
INSUP LEE, University of Pennsylvania

Web-based malware is a growing threat to today's Internet security. Attacks of this type are prevalent and lead to serious security consequences. Millions of malicious URLs are used as distribution channels to propagate malware all over the Web. After being infected, victim systems fall in the control of attackers, who can utilize them for various cyber crimes such as stealing credentials, spamming, and distributed denial-of-service attacks. Moreover, it has been observed that traditional security technologies such as firewalls and intrusion detection systems have only limited capability to mitigate this new problem.

In this paper, we survey the state-of-the-art research regarding the analysis of — and defense against — web-based malware attacks. First, we study the attack model, the root-cause, and the vulnerabilities that enable these attacks. Second, we analyze the status quo of the web-based malware problem. Third, three categories of defense mechanisms are discussed in detail: (1) building honeypots with virtual machines or signature-based detection system to discover existing threats; (2) using code analysis and testing techniques to identify the vulnerabilities of web applications; and (3) constructing reputation-based blacklists or smart sandbox systems to protect end users from attacks. We show that these three categories of approaches form an extensive solution space to the web-based malware problem. Finally, we compare the surveyed approaches and discuss possible future research directions.

Categories and Subject Descriptors: D.4.6 [**Security and Protection**]: Invasive software (*e.g.,* viruses, worms, Trojan horses); C.2.0 [**Computer-Communication Networks**]: General—*Security and protection (*e.g., *firewalls)*

General Terms: Security, Verification, Design

Additional Key Words and Phrases: Web-based malware

1. INTRODUCTION

The Web has become the core of our daily computing experience. Web applications and services have been developed and deployed with unprecedented speed, providing various important functionalities to the end user such as office applications, social networking, content sharing, education, and entertainment. From 2005 to 2008, the number of indexable webpages has grown from a few billion to a trillion [Gulli and Signorini 2005; Google Web Index 2008]. In 2010, the population of Internet users was about two billion [Internet Stats 2010]. Moreover, these numbers are rapidly increasing.

Given its popularity and ubiquity, the Web also attracts the attention of malicious entities. Indeed, the Web and its global user community have observed various forms of attack in the past [Rubin and Geer 1998; Garfinkel and Spafford 2001]. Among these attacks, using the Web as a channel to distribute malware (*i.e.,* malicious software) has become a prominent

This work was supported in part by ONR MURI N00014-07-1-0907. POC: Insup Lee, lee@cis.upenn.edu
Author's address: J. Chang, K. K. Venkatasubramanian, A. G. West and I. Lee, Department of Computer and Information Science, University of Pennsylvania, 3330 Walnut Street, Philadelphia, PA 19104.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

issue. This issue has generated a great deal of attention from both the security research community and the general public. Web-based malware is disseminated when victim users visit malicious websites. This malware is designed to conduct various cyber crimes, such as gaining control of the victim system, stealing private information, launching denial-of-service attacks, and spamming. According to the latest report from Dasient, a web security solutions vendor, the number of websites that deliver malware has doubled in one year, between 2009 and 2010 [Dasient Report 2010].

The arms race between attackers and defenders has led to the adoption of increasingly sophisticated malware distribution tactics. The traditional malware distribution model is *push-based* — attackers actively search for and infect the victim systems. However, security technologies such as firewalls have been widely deployed in enterprise networks, which provide good defense against techniques that were commonly used for push-based propagation. As a result, the malware distribution has evolved to a *pull-based* model, where victims unknowingly visit malicious websites. Under the pull-based model (*i.e.,* web-based malware distribution), the channel for attack is initiated and established by potential victims, which significantly lowers the defense barrier for attackers to cross.

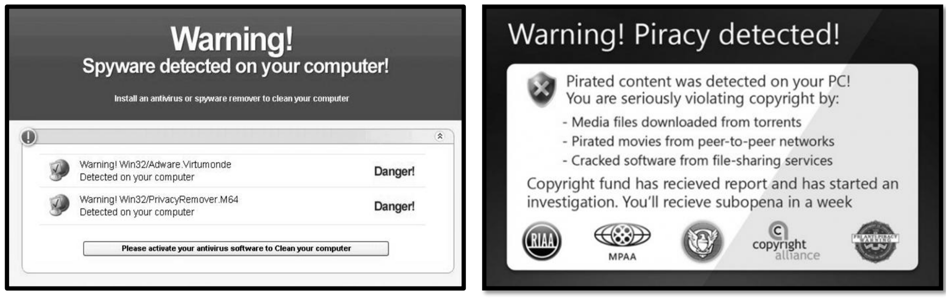
Defending against web-based malware is a difficult task. First of all, as the web browser becomes the battlefield, the defense boundary is very long — not only billions of active personal computers, but also portable devices with web browsing functionality can become the target of the attack [Schmidt et al. 2009; Fleizach et al. 2007]. Second, typical web users may not have enough knowledge and expertise to protect themselves, making them potentially the weakest link in the entire defense chain. Third, since the Web is a huge distributed system, there is no single trust authority, making the defense strategy inherently collaborative. A global defense is infeasible in practice given that certain parts of the Web have been known to be miscreant-friendly [RBN Study 2007]. Finally, the bar for conducting effective web attacks is rather low; key techniques and vulnerabilities of victim devices are well-documented [Daniel et al. 2008]. Automatic tools for facilitating the attacks are also easily obtainable [Binsalleeh et al. 2010; Ormerod et al. 2010].

Recent trends in web application development further complicate the problem. The current generation of Web applications (*i.e.,* Web 2.0) concentrates on presenting user generated content. Although the openness of these applications provides useful functionalities to the end user, at the same time, it opens holes for attacks [Lawton 2007]. Moreover, a popular web programming paradigm known as the *mashup* has been adopted to quickly develop new applications by reusing existing content [Sabbouh et al. 2007]. However, this trust of external data and code is blind and creates new vulnerabilities [Tipton 2009; Magazinius et al. 2010]. For instance, a huge number of existing web sites depend on the income from advertisements, which are provided by third-party ad syndications. It has been observed that many legitimate websites are involved in delivering malware due to unsafe advertisements [Provos et al. 2008].

1.1. Outline

Considering all of the challenges discussed above, it is clear that understanding the web-based malware problem and developing effective solutions are urgent tasks for the security of the Internet. Indeed, great effort has been made by security researchers, who are attempting to shed light on this issue. This work analyzes many different aspects of the problem and provides insights for designing systematic defense strategies. In this paper, we present a survey of the literature, which is organized into four main topics:

(1) THE STATUS QUO OF WEB-BASED MALWARE: Section 3 presents an analysis of the current status of web-based malware problems that are measured by Internet-scale monitoring; this analysis illustrates the scope of this issue.



Warning!
Spyware detected on your computer!
Install an antivirus or spyware remover to clean your computer

Warning! Win32/Adware.Virtumonde
Detected on your computer
Danger!

Warning! Win32/PrivacyRemover.M64
Detected on your computer
Danger!

Please activate your antivirus software to Clean your computer

Warning! Piracy detected!

Pirated content was detected on your PC!
You are seriously violating copyright by:
- Media files downloaded from torrents
- Pirated movies from peer-to-peer networks
- Cracked software from file-sharing services

Copyright fund has recieved report and has started an investigation. You'll recieve subopena in a week

Fig. 1: Screen-shots of Social Engineering Tactics

(2) ATTACK DETECTION: Section 4 surveys the approaches that detect existing web-based malware attacks. These approaches help researchers have a better understanding of the problem in order to design effective countermeasures.
(3) VULNERABILITY IDENTIFICATION: Section 5 surveys code analysis techniques that are designed to minimize the number of vulnerabilities that exist in web applications. These vulnerabilities are often used by attackers to turn legitimate websites into a part of the malware distribution infrastructure.
(4) ATTACK PROTECTION: Section 6 surveys the defense techniques that protect victim systems from being infected. These approaches are effective for mitigating the impact of existing attacks.

Representative approaches surveyed in this paper are shown in Table II. Section 7 presents a detailed discussion of the surveyed work regarding its ideal deployment scale, comparative effectiveness, degree of autonomy, adaptiveness, and gamesmanship. In Section 8, we conclude the paper with possible future research directions. Before delving into the technical details, the background and the attack model of web-based malware are presented and analyzed in Section 2.

2. BACKGROUND

In this section, we first formulate the attack model of web-based malware. Then, we discuss the root causes and vulnerabilities that enable web-based malware attacks. Approaches proposed in the literature to address the problem from various aspects are highlighted. Finally, related survey studies are presented and briefly discussed.

2.1. Terminology

To understand web-based malware, it is important to carefully define the terms that we use throughout this paper. *Malware*, which is short for "malicious software", is software designed to access the resources of a computer system without the owner's informed consent [Christodorescu et al. 2007]. Malware includes computer viruses, worms, spyware, and other malicious and unwanted software. A *web-based malware attack* is defined as the process of malware downloading and installing on a victim computer system through visiting infected landing web sites. In this paper, we use the terms "web-based malware attack", "web-based malware", and "attack" interchangeably. Similar to the definition in Provos et al. [2008], the term *landing web site* or *landing site* is defined as the website that initiates web-based malware attacks, when victim users visit it. On the other hand, websites that host malicious shellcode or malware used in the attack, are termed *malware distribution sites* or *distribution*

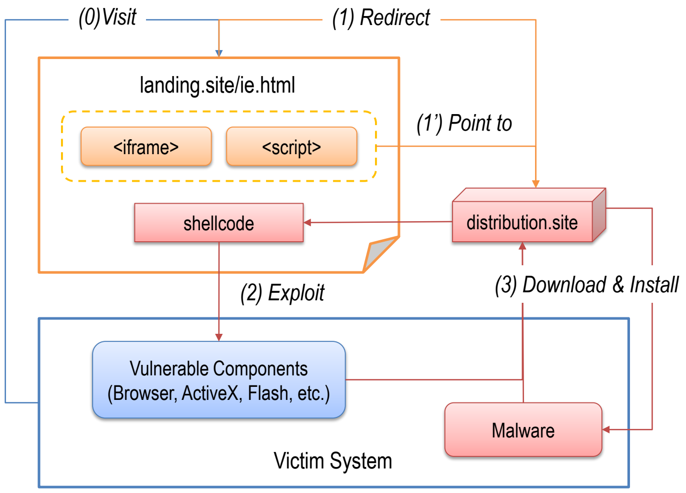


Fig. 2: Drive-by Download Attack Scenario

sites. Detailed discussion on the relationship between landing sites and distribution sites is presented in Section 3.2.

According to IETF RFC 2828, a *vulnerability* of a computer system is defined as a flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy [RFC-2828 2000]. A *zero-day vulnerability* is defined as a vulnerability without an effective patch. *Shellcode* (also known as an *exploit*) is a small piece of computer software that takes advantage of a vulnerability to cause unintended behavior of a computer system, which is commonly used to distribute malware.

2.2. Attack Model

There are two types of techniques used by attackers to perform web-based malware attacks, namely: (1) *Social Engineering*: using psychological manipulations and decoys to trick the victims into authorizing the downloading and installation of malware; and (2) *Drive-by Download*: designing a web page that contains malicious code to trigger the downloading and installation of malware automatically.

The social engineering technique often involves using alluring descriptions and enticing a victim to make decisions that compromise his computer-security. Screen-shots of real social engineering attacks are illustrated in Figure 1. Social engineering tricks are designed to exploit the cognitive biases of the human decision-making process rather than exploiting the vulnerabilities of the victim computer systems. For example, the malicious webpage can alert the victims with fake virus detection messages and ask them to download rogue anti-virus software [Cova et al. 2010], or display fake award notifications for victims to collect by clicking `URL`s linked to malware [Sidiroglou et al. 2005].

On the other hand, the drive-by download attack may form a larger threat to victims given its automated nature. Therefore, it also requires more careful study to understand the attack mechanism. In general, the drive-by download malware attack involves three major steps as shown in Figure 2:

(1) *Fetch and Prepare Malicious Shellcode:* In this step, shellcode used for exploiting a victim system is fetched from the distribution site through a visit to the landing site. This is usually achieved in two ways: (1) by placing `script` or `iframe` tags on the

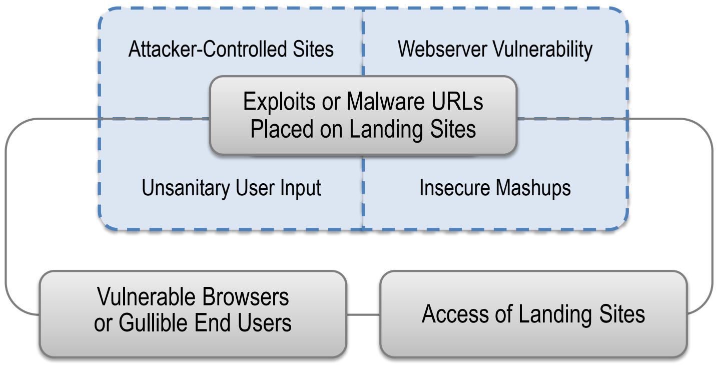

Attacker-Controlled Sites
Webserver Vulnerability
Exploits or Malware URLs Placed on Landing Sites
Unsanitary User Input
Insecure Mashups
Vulnerable Browsers or Gullible End Users
Access of Landing Sites

Fig. 3: Necessary Conditions of Web-based Malware Attack

landing site with `src` attributes pointing to the distribution site or (2) by redirecting the browser window through various mechanisms (*e.g.,* HTTP 302 temporary redirect or JavaScript APIs). The malicious code being fetched is often highly obfuscated to complicate investigation and to escape signature-based security scanners [Feinstein and Peck 2007]. In order to increase the probability of a successful attack, the malicious code often checks the environment of the victim system, such as the browser type, available plugins, or version information. This enables the attacker to find all available vulnerabilities to exploit. As observed in Wang et al. [2006], attackers often target multiple vulnerabilities during one attack in order to increase their chance of success.

(2) *Exploit Vulnerability:* The shellcode that attackers want to execute is injected by exploiting browser vulnerabilities. Various techniques (*e.g.,* heap overflow [Daniel et al. 2008; Ratanaworabhan et al. 2009]) can be used to hijack the execution flow of the browser or its components (*e.g.,* ActiveX control, Flash player, PDF helper object). Most of the techniques are based on the idea of conducting clever memory allocations through the manipulation of a number of JavaScript strings [Sotirov 2007; Sotirov and Dowd 2008]. Regardless of the technique used by the attackers, the shell code is designed to initiate the download and installation of binaries on the victim system to achieve a larger attack goal.

(3) *Download and Install Malware:* The last step of the attack is to download malicious binaries from a remote malware distribution site and install them on the victim system. A plethora of malware types have been observed in past attack instances, including viruses, backdoors, Trojan downloaders, and spyware [Wang et al. 2006]. Typically, victim machines will become parts of a botnet and used as resources to perform other types of cyber crime, such as distributed denial-of-service attacks (DDoS), or spamming [Polychronakis et al. 2008].

2.3. Root Causes and Vulnerabilities

As we discussed above, there are **three necessary conditions** for a successful web-based malware attack, as illustrated in Figure 3: (1) the presence of vulnerable web browsers or end users who are gullible to social engineering tricks, (2) effective exploits or `URL`s pointing to malware present on landing sites, and (3) the access of landing websites. The first condition regarding software vulnerability and human decision-making weakness has been studied extensively in computer security research. The third condition is the *pull-based* malware propagation model, which is straightforward. In this section, we primarily focus on studying the second necessary condition. The following question can be reasonably raised: How are exploits or malware `URL`s placed on a landing site?

One possible answer is that all the landing sites are operated by the attackers themselves. Then it would be trivial for attackers to place the necessary exploits or URLs. However, a recent study has shown that it is not the case for most of the real-world attacks, since a large portion of the landing sites are in fact legitimate sites being infected [Provos et al. 2008]. Here, we identify and discuss three main causes that lead to infections:

(1) *Webserver Vulnerability:* If an attacker is able to gain control of a webserver, he can modify the web content hosted on the server for his own benefit. A sophisticated webserver usually has a large set of software installed, and the webserver is only as secure as its weakest component. Operating systems and application software used by webservers are often found to contain vulnerabilities. This makes attacks on webservers feasible and dangerous. For example, BIND vulnerabilities [1998] would allow attackers to gain root privileges on a webserver system. This type of attack is particularly damaging to large virtual hosting farms. As observed in Provos et al. [2007], it can turn the whole web hosting infrastructure into a large malware distribution center.
(2) *Unsanitary User Input:* Many web applications allow users to contribute content [Lin 2007]. Apart from being an important functionality to normal users, it may also open doors for attackers to place malicious content. Without a careful sanity check, content provided by attackers might be presented directly to other users of the same web application (*e.g.,* cross-site scripting attacks), or be used by the web application itself in security-critical operations (*e.g.,* SQL injection attacks). Such security vulnerabilities lie in the code of web applications. A typical web application often has code executing at either the server side (*e.g.,* ASP.NET, JSP, PHP) or the client side (*e.g.,* JavaScript, Flash). Efforts have been made to check vulnerabilities of the server-side code, and thousands of such vulnerabilities are reported every year [Bau et al. 2010]. However, as demonstrated in a recent study, insecure client-side coding practices are also prevalent [Yue and Wang 2009]; these practices favor the attackers.
(3) *Insecure Mashup:* In web development, a mashup is a hybrid web application that combines content from two or more sources to create new services [Crites et al. 2008]. Concrete examples of mashup often include using advertisements from third-party ad syndication, widgets, and code libraries. As a popular web development paradigm, mashup implies trust on external code or data. However, as the external content is usually delivered to a user's browser directly from the external source, mashup builders have very limited control over it. Indeed, the current practice of web mashup has been problematic, and it has become a major channel to infect legitimate sites for conducting web-based malware attacks [Provos et al. 2008; Cova et al. 2010].

2.4. Existing Defense Approaches

With the root causes of the web-based malware problem in mind, in this section, a overview of existing defense approaches is presented with brief discussions on how these approaches operate by addressing one or more of the root causes.

2.4.1. General Defense Approaches. A Web-based malware attack is not greatly different from a traditional push-based malware attack, except for its distribution channel and propagation model. Therefore, general defense mechanisms (see Table I) against traditional malware can be adopted to mitigate the problem.

(1) *Using the Most Updated Browser and Plugins:* This approach attempts to eliminate the first necessary condition (see Section 2.3) by minimizing the number of vulnerabilities that can be exploited by an attacker. Specifically, this approach works by shortening the lag between the time of patch release and the time of patch installation by the end user. Most popular web browsers and their plugin components have built-in upgrade mechanisms to facilitate this process. However, as demonstrated in Wang et al. [2006],

Approach	Strength	Weakness
Updated Browser	High availability; built-in upgrade support; effective for known vulnerabilities.	Ineffective for zero-day vulnerabilities; significant lag and risk.
Anti-Virus Software	Well maintained malware definition databases are available; low false positives for known binaries.	High false negatives for newly observed binaries; system performance may be slower due to scanning overhead.
Sandbox	Adaptive to different attacks by setting proper security policies.	Learning curve to properly configure the corresponding policies is considerable.

Table I: Strengths and Weaknesses of General Defense Approaches

many landing sites actively use zero-day exploits to attack victim systems, and sophisticated attackers keep discovering new exploits to improve their probability of success. It sometimes takes a significant amount of time for the software vendors to release patches for newly discovered vulnerabilities. Therefore, using the most updated software is not enough to defend against web-based malware attack.

(2) *Using Anti-virus Software:* Since the main objective of the attack is to download and install malware on the victim machine, anti-virus tools can be used to detect and remove the binaries that are covertly fetched. If the anti-virus engine can correctly identify all types of malware, this approach can form a powerful defense. Unfortunately, the state-of-the-art anti-virus techniques still depend on certain forms of signature matching and need to update the malware definition continuously to be effective. Arguably, the malware definitions can never be complete, as malware keeps evolving over time. Indeed, Provos et al. [2008] has shown that even the best anti-virus engine on the market (armed with the latest definitions) can only achieve an average detection rate of 70%. Therefore, this approach alone is not sufficient either.

(3) *Using Sandbox for Untrusted Software:* A *sandbox* is a well-studied security mechanism for running untrusted programs. Typically, the sandbox provides an environment with limited resources for untrusted programs to run in. Well-known examples of sandboxing include virtual machines, capability systems, and the rule-based execution framework. The main drawback of the sandbox mechanism is the difficulty of its policy configurations, *i.e.,* which programs should be executed inside the sandbox, and what privileges should be given to the untrusted software. However, common computer users often lack the expertise to set such security policies properly. For instance, malware delivered by social engineering tricks may entice a user to make incorrect decisions that compromise his own computer security.

2.4.2. Specific Defense Approaches. Realizing the insufficiency of the general defense mechanisms, many approaches have been proposed that focus on addressing the special characteristics of the web-based malware problem. Such approaches can be classified into three categories, and we survey representative approaches for each category as shown in Table II. Table III describes the necessary conditions (see Section 2.3) that these approaches attempt to break, and the corresponding deployment location:

Approach		Other Work
Detect Attack Vector Presence on the Web (DA) (See Section 4)	VM-based	Wang et al. 2006; Moshchuk et al. 2006; Provos et al. 2007; Provos et al. 2008; Polychronakis et al. 2008;
	Signature-based	Roesch 1999; Toth and Kruegel 2002; Akritidis et al. 2005; Polychronakis et al. 2007; Nazario 2009; Ratanaworabhan et al. 2009; Curtsinger et al. 2010; Song et al. 2010; Cova et al. 2010
Identify the Vulnerabilities of Web Applications (IV) (See Section 5)	Server-side	Huang et al. 2004; [Jovanovic et al. 2006a]; Xie and Aiken 2006; Wassermann and Su 2007; Lam et al. 2008; Balzarotti et al. 2008; Bau et al. 2010; Balduzzi et al. 2011
	Client-side	Chugh et al. 2009; Guarnieri and Livshits 2009; Saxena et al. 2010 (FLAX); Saxena et al. 2010 (Kudzu); Chang et al. 2011
Protect Victims from Attacks (PA) (See Section 6)	Blacklist	Ma et al. 2009; Antonakakis et al. 2010
	Sandbox	Cox et al. 2006; Jain et al. 2008; Grier et al. 2008; Reis and Gribble 2009; Lu et al. 2010; Li et al. 2011

Table II: Taxonomy of Surveyed Work

(1) *Detect Attack Vector Presence on the Web (DA):* Approaches in this category attempt to detect existing attacks to facilitate further study of the problem. By detecting existing attack vector presence, researchers can discover new exploitation techniques and zero-day vulnerabilities used by attackers. Two types of approaches are surveyed in this regard: (1) building a high-interaction honeypot using a virtual machine (VM), and (2) building signature-based attack detection systems.
(2) *Identify the Vulnerabilities of Web Applications (IV):* Approaches in this category address the second necessary condition by conducting effective code analyses to discover web application vulnerabilities. As a result, the chance of legitimate websites becoming malicious landing sites decreases significantly. Two types of approaches are surveyed in this paper: (1) using code analysis or testing to identify server-side component vulnerabilities, and (2) using static analysis or hybrid techniques to discover client-side code vulnerabilities.
(3) *Protect Victims from Attacks (PA):* Approaches in this category attempt to mitigate the impact of existing attacks. This is achieved by countering the third necessary condition. Two approaches[1] are surveyed in this paper: (1) designing smart sandbox mechanisms that nullify the functionality of the malware downloaded; and (2) calculating the reputation of malware landing sites to construct dynamic blacklists, which can then be used by end users to avoid visiting dangerous sites.

[1]The attack detection techniques can also be extended to act as protection mechanisms by dissuading the end users from visiting the sites with known malware presence. Discussion is presented in Section 6.

Approach		Target Necessary Condition	Deployment
DA	VM-based	Condition 3: victim visiting the landing sites to be infected.	Third party
	Signature-based		
IV	Server-side	Condition 2: effective exploits or `URL`s linked to malware being placed on the landing sites.	Web Server
	Client-side		
PA	Blacklist	Condition 3: victim visiting the landing sites to be infected.	Client System
	Sandbox	Condition 1: vulnerable web browsers or components being used by the victim.	

Table III: Comprehensiveness of Surveyed Work

These three categories cover the cutting-edge research regarding the web-based malware problem. Together, these approaches form an extensive solution space by addressing all the necessary conditions of web-based malware attacks. Before delving into the details of these approaches, we present the status quo of web-based malware based on Internet-scale monitoring, which illustrates the severity of the issue and provides useful insight into the problem.

2.5. Related Work

To our best knowledge, this paper is the first comprehensive survey focusing on web-based malware analysis and defense. A summary comparison of this paper and related work is illustrated in Figure 4. Christodorescu et al. [2007], Vinod et al. [2009] and Idika and Mathur [2010] focus on understanding the state-of-the-art of malware detection techniques in general. These studies compare three major classes of malware detection (*i.e.,* behavior-based, signature-based and specification-based) and malware code obfuscation techniques. Further, Jacob et al. [2008] and You and Yim [2010] present a more detailed survey on behavior-based malware detection and malware code obfuscation techniques, respectively. Siddiqui et al. [2008] and Shabtai et al. [2009] survey the adoption of machine-learning approaches to detect malware using file features. All these studies advance the understanding of building accurate and robust malware detection tools, which are considered general defense mechanisms against web-based malware as discussed in Section 2.4.1. Qing and Wen [2005] and Li et al. [2008] studied the traditional push-based malware propagation model used by Internet worms and the research efforts on the detection and containment of such cyber attacks. Combining our work and the useful insights provided by these studies, one can form a better view of how the malware propagation model evolved over time.

Considering the web-based malware attack as a generic threat to system security, studies such as Lunt [1993], Mukherjee et al. [1994], Bai and Kobayashi [2003], Murali and Rao [2005], Sadoddin and Ghorbani [2006], Marhusin et al. [2008], Sabahi and Movaghar [2008] and Hosseinpour et al. [2010] shed light on the understanding of generic system intrusion issues and the countermeasures. Focusing on the botnet threats and detection techniques, studies Peng et al. [2007], Zhu et al. [2008], Li et al. [2009], Bailey et al. [2009], Feily et al. [2009] and Zeidanloo et al. [2010] extend our discussion on the post malware infection

		This Paper	Related Work				
			Christodorescu et al. 2007; P. et al. 2009; Idika and Mathur 2010	Jacob et al. 2008; Siddiqui et al. 2008; Shabtai et al. 2009; You and Yim 2010	Qing and Wen 2005; Li et al. 2008	Lunt 1993; Mukherjee et al. 1994; Bai and Kobayashi 2003; Murali and Rao 2005; Sadoddin and Ghorbani 2006; Marhusin et al. 2008; Sabahi and Movaghar 2008; Hosseinpour et al. 2010	Peng et al. 2007; Zhu et al. 2008; Li et al. 2009; Bailey et al. 2009; Feily et al. 2009; Zeidanloo et al. 2010
Scope	Web-based Malware	✓					
	General Malware		✓	✓			
	Internet Worm				✓		
	System Intrusion					✓	
	Botnet						✓
Propagation Model	Push-based		✓	✓	✓	✓	✓
	Pull-based	✓					
Surveyed Approach	Attack Detection	✓	✓	✓	✓	✓	✓
	Vulnerability Identification	✓	✓				
	Attack Protection	✓	✓		✓	✓	
	Post-Attack Recovery						✓

Fig. 4: Summary Comparison of Related Work

consequences in Section 3.3. These two categories of studies complement our work and offer different viewpoints for cogitating the entire network security ecosystem.

3. THE STATUS QUO OF WEB-BASED MALWARE

From Jan. 2007 to Oct. 2007, 66 millions `URL`s were examined in-depth to discover existing web-based malware attacks [Provos et al. 2007; Provos et al. 2008; Polychronakis et al. 2008]. This Internet-scale measurement built a representative corpus to study the current status of the web-based malware problem. An illustration of the studied web-based malware infrastructure is shown in Figure 5. Techniques adopted by this study to detect web-based malware are very similar to the approach proposed in Wang et al. [2006] (see Section 4.1). Here, we first present the observations made in this study.

3.1. Attack Prevalence

— *Drive-by download attacks are prevalent, and are becoming increasingly common.* During the study phase, over 3 million `URL`s were found pointing to webpages that conducted effective drive-by download attacks. These malicious `URL`s correspond to 180,000 landing sites, and 9,000 malware distribution sites.

— *The risk of users being exposed to an attack is significant.* The malicious `URL`s are uniformly distributed over different website functional categories (*e.g.,* news, sports, arts). Moreover, 1.3% of search queries to Google returned at least one malicious `URL`, and this number increased linearly by 0.1% every month. Further, 0.6% of the top million `URL`s that appear most frequently as Google's search results were malicious.

3.2. Distribution Infrastructure Analysis

— *Weak security practices are prevalent on landing site webservers.* A large fraction of the landing site servers were running outdated software with documented vulnerabilities. For example, 38.1% of Apache web servers and 39.9% of servers with PHP scripting support reported using an old version of software with known security vulnerabilities.

— *Web advertisements are an significant channel to deliver attacks.* On average, 2% of the landing sites and 12% of landing pages were malicious due to unsafe advertisements. Ad-delivered attacks have three special characteristics: (1) malware delivered via ads have longer chains between landing sites and distribution sites due to multiple levels of ad syndication; (2) certain ad syndications have been observed to deliver malware directly (instead of being intermediate nodes); and (3) attacks appear in short-lived spikes — many landing sites are affected simultaneously by the same infected ad syndication. But once detected by the ad provider, the malicious content is removed very quickly.

— *Strong locality has been observed.* In terms of geography, a handful of countries were hosting around 85% of all the landing and distribution sites. Moreover, the landing sites and the corresponding distribution sites are highly localized within the same geographic boundaries. In terms of network, a small IP range was found to host up to 210 distribution sites. 70% of all the malicious sites have IP addresses belong to four /8 prefixes. Further, 98% of these sites were hosted by only 210 Autonomous Systems (out of 40,000 active ASes).

— *The topology of the malware distribution infrastructure has interesting properties.* The distribution-versus-landing site ratio is a long-tail distribution: a large percentage (around 45%) of the distribution sites had only one landing site; on the other end of the curve, some distribution sites had more than 21,000 associated landing sites. This distribution pattern can be explained by attackers making a trade-off between the ease of management and the detection avoidance. Interestingly, 80% of the distribution sites share at least one landing site with each other.

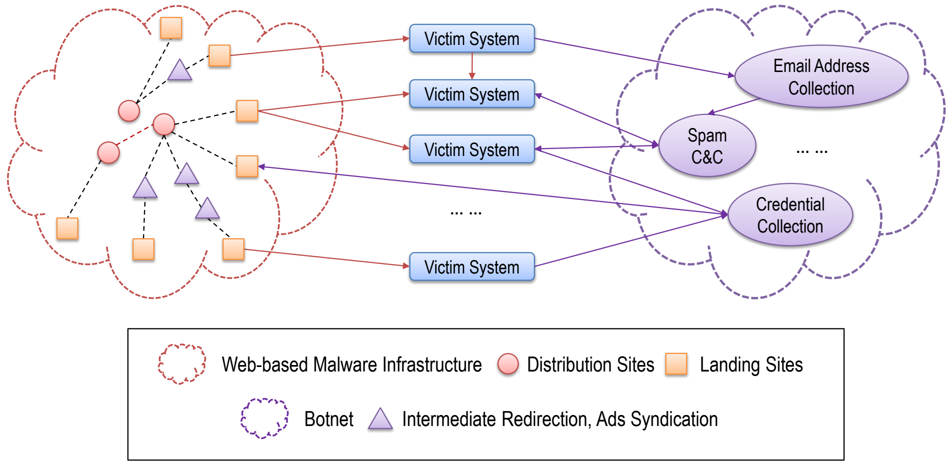


Fig. 5: Example of Web-based Malware Infrastructure

3.3. Malware and Post Infection Analysis

— *Web-based malware has great diversity and keeps evolving.* 3% of the distribution sites hosted more than 100 different binaries. Adware, Trojans and computer worms are the top three malware types on the web. Moreover, these malware binaries keep changing with a frequency ranging from once every few hours to once every several days. Some malicious URLs have been observed changing the corresponding binaries almost every hour. This dynamism could explain the insufficiency of the signature-based anti-virus technique for detecting web-based malware.

— *Web-based malware seriously threatens a victim's system security.* A successful attack leads to on average 8 different malware binaries downloaded into the victim system. In 7% of the cases, victim machines had the new Browser Helper Object (BHO)[2] installed, 23.5% had the browser home page or the name server changed, 36.2% had the firewall and other security settings changed, and 51.3% had the system startup setting modified in order to have malware persist across reboots.

— *Web-based malware provides a cornerstone for large-scale electronic crime.* After installation on the victim system, malware notifies attackers about the compromised system and the sensitive exfiltrated information. Many common Windows protocols (*e.g.,* NETBIOS, SMB, DCOM, MSSQL), are also probed by malware to scan for the vulnerabilities of neighboring computers to propagate further. Moreover, various communication protocols (*e.g.,* HTTP, IRC) are used by malware to join botnets for conducting other cyber crimes (*e.g.,* Denial-of-Service, spamming).

3.4. Related Measurement Studies

Prior to this Internet-scale measurement study, several relatively small-scale studies were conducted. A summary comparison of such studies is shown in Table IV. Wang et al. [2006] examined 17,000 URLs and found about two hundred of them to be malicious in nature. In this study, multiple zero-day vulnerabilities were also discovered and studied. In May 2005, 18 million URLs were crawled to study a specific type of malware — spyware — on the web [Moshchuk et al. 2006]. Using a security scanner, 45,000 URLs were found to be linked to spyware. Further, the author observed that different exploit mechanisms were used to

[2]BHO is a type of plugin for Internet Explorer browser to provide extra functionality.

	Paper			
	Provos et al. 2008;	Wang et al. 2006	Moshchuk et al. 2006	Bailey et al. 2007
Study Scope	Web-based Malware	Web-based Malware	Web-based Spyware	Malware
Study Scale	Billions of `URLs`	17,000 `URLs`	18 million `URLs`	3698 pieces of binaries
Approach	VM-based Detection	VM-based Detection	VM-based Detection	Anti-virus Software

Table IV: Summary Comparison of Measurement Studies

target different browser platforms. It is important to note that the detection techniques used by these papers can suffer from false negatives, which make their measurement result a conservative estimate of the real situation. Therefore, all these works emphasize the trend of web-based malware attacks as a rising threat for the Internet.

Bailey et al. [2007] focuses on achieving better understanding on the nature of malware spreading over the Internet, which is orthogonal to the studies discussed above. Motivated by the observations that host-based anti-virus software provides inconsistent information about the underlying threats, the authors design a behavior-based classification technique to automatically categorize malware. By evaluating the proposed scheme over 3698 pieces of malware collected over 6 months, this behavior-based classification technique demonstrates better completeness and accuracy than existing approaches. This work can benefit the post infection analysis of web-based malware attacks.

4. WEB-BASED MALWARE DETECTION

As we saw in Section 3, to gain a better understanding of the web-based malware problem, it is necessary to detect the instances of existing attacks. Given the huge number of webpages on the Internet and their rapidly evolving nature, manual inspections are clearly insufficient. Therefore, techniques have been proposed in the literature to automate this process. Such approaches can be categorized into two types — virtual machine-based and signature-based — which are effective for detecting web-based malware delivered through drive-by download attacks. A brief summary of surveyed work in this section is shown in Table V.

4.1. Virtual Machine-Based Detection

In Wang et al. [2006], the authors proposed an automatic detection system named Honey-Monkey, which follows a black-box approach and detects drive-by download attacks using virtual machines (VMs). The system is given the name HoneyMonkey for two reasons: (1) the system is a “honeypot”, which attracts attacks, and (2) a computer program (monkey) is designed to simulate the web browsing activities of a human and to detect attacks.

The system design of HoneyMonkey is shown in Figure 6. Two types of VM are used in the HoneyMonkey system. Type I: unpatched virtual machines with known vulnerabilities; and Type II: fully patched virtual machines that are only vulnerable to zero-day exploits. Both types are designed to run the Microsoft Windows operating system. Before running

Category	Paper	Contribution Summary
VM-Based	Wang et al. 2006	Using VM to effectively detect web-based malware attacks and zero-day vulnerabilities.
	Moshchuk et al. 2006; Provos et al. 2008	Enabling efficient VM-based detection of web-based malware attacks over large-scale dataset.
Signature-Based	Cova et al. 2010	Using an instrumented web browser to extract effective signatures to detect drive-by download attacks.
	Song et al. 2010	Extracting signature of browser communication behavior to detect drive-by download attacks.
	Curtsinger et al. 2010	Extracting syntax signature of malicious JavaScript to detect drive-by download attacks.
	Roesch 1999; Nazario 2009	Matching data-flow signature at network or application level to detect potential system intrusions.
	Toth and Kruegel 2002; Akritidis et al. 2005; Polychronakis et al. 2007; Ratanaworabhan et al. 2009	Using memory overflow signatures to detect buffer overflow attacks.

Table V: Summary of Web-based Malware Detection Techniques

HoneyMonkey for detection, a set of candidate `URL`s needs to be collected; these `URL`s are then used as the input of the HoneyMonkey system. HoneyMonkey works in three stages:

(1) Stage-1 HoneyMonkey visits N number of `URL`s simultaneously by the monkey program in a Type I VM instance. A time-out is set to properly load all the `URL`s in the VM. By closely monitoring a group of persistent-state changes in the VM (*e.g.,* executable file creation, Windows registry modification, *etc.*), the program decides on whether further investigations are needed. That is, if no persistent-state change is found (*i.e.,* the VM state is *clean*), then the system fetches the next N `URL`s from the candidate `URL` pool and restarts Stage 1. Otherwise, the HoneyMonkey system will switch to Stage 2.
(2) Stage-2 HoneyMonkey pinpoints a subset of the N `URL`s that trigger the persistent-state changes. To achieve this, only one `URL` is loaded in a Type I VM every time. By combining Stage 1 and 2, the system can reuse a clean VM instance for a rather large number of `URL`s without sacrificing accuracy. When a malicious `URL` that triggers persistent-state changes is discovered in Stage 2, it is further examined in Stage 3.

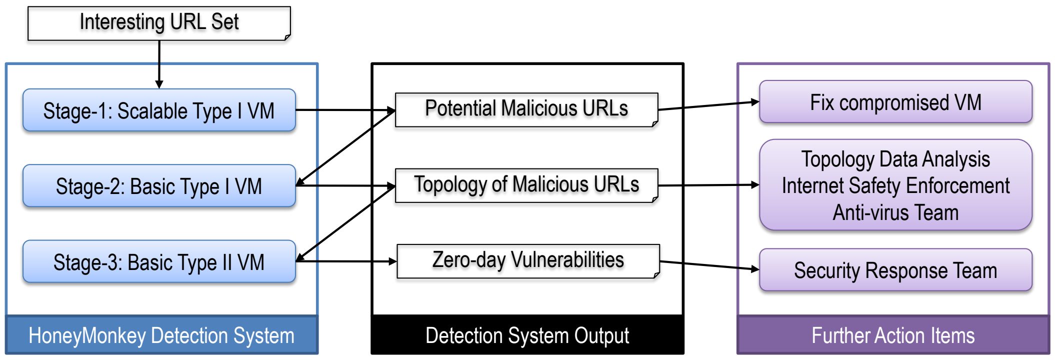


Fig. 6: HoneyMonkey System Design

(3) Stage-3 HoneyMonkey reloads the malicious `URL` in a Type II VM to discover zero-day exploits.

Five pieces of information are recorded during the detection using various techniques [Wang et al. 2003; Beck et al. 2005; Joshi et al. 2005]: (1) file operations, (2) the process creation log, (3) the Windows registry modification log, (4) exploited vulnerabilities, and (5) browser redirection activities. Careful analyses of the recorded information are used to understand the attack techniques and the malware distribution infrastructure. Within the first month of system deployment, 752 malicious `URL`s and 1 zero-day exploit were discovered by HoneyMonkey, proving its effectiveness.

Moshchuk et al. [2006] and Provos et al. [2008] improved the basic VM-based detection system by adding a pre-processing phase. During the pre-processing, webpages are checked using static code analysis techniques or various heuristics (*e.g.,* whether the webpages contain `URL`s linked to known attacking sites or heavily obfuscated JavaScript code) to filter out a large portion of benign webpages and reduce the cost of launching VM instances. This improvement enables the efficient examination of candidate webpages at large-scale (*i.e.,* from millions to billions), using VM-based detection systems.

4.2. Signature-Based Detection

Using VMs as high-interaction honeypots has its own challenges. In particular, an attack can be detected only if the vulnerabilities that are being exploited by the attackers are present in the honeypot system. However, since there are so many different versions of web browsers and hundreds of components (*e.g.,* PDF plugins, Flash, and ActiveX controls) have been targeted to launch web-based malware attacks, it would be very difficult to configure any one system as being the most "vulnerable" to detect all possible attacks. Otherwise, the false negative rate would be rather high as observed in Seifert et al. [2009].

In Cova et al. [2010], the authors proposed an alternative approach to build high-interaction honeypots. Their approach is named JSAND, which combines the usage of an instrumented browser, a set of effective signature features, and machine learning techniques to detect drive-by download attacks. The instrumented browser is designed to facilitate three tasks: (1) to simulate the personality of various browsers (*e.g.,* Internet Explorer, Firefox, Chrome) to expand the available attack surface; (2) to simulate arbitrary system environments and configurations. For example, requests for loading ActiveX controls or plugins can be fully customized; and (3) to track all JavaScript function definitions and invocations for implementing anti-cloaking mechanisms [Moser et al. 2007; Wilhelm and Chiueh 2007].

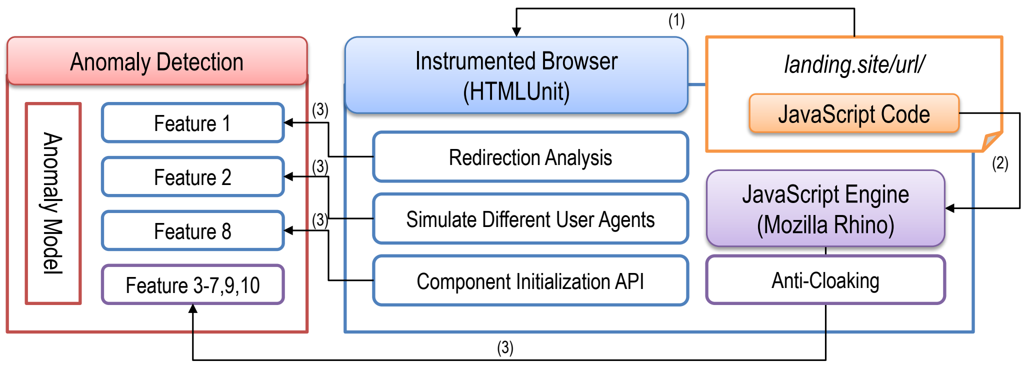


Fig. 7: JSAND System Design

The overall design of the JSAND system is illustrated in Figure 7. Ten signature features are identified and used in training the anomaly model of drive-by download attacks. The authors argue that these ten signature features are effective to distinguish between benign webpages and malicious ones:

— *Feature 1: the length and target of the redirection chain* — it has been observed that malicious landing sites often have unusually long redirection chains toward malware distribution sites.
— *Feature 2: the browser personality and history-based differences* — as many attacks only target a specific platform (*e.g.,* Internet Explorer 6 running on Windows XP), the landing websites might return different content accordingly.
— *Feature 3 to 5: the ratio of string definitions and string uses, the number of dynamic code executions, and the length of dynamically evaluated code* — these features are included to capture the characteristics of code obfuscation techniques, which are often used by malicious sites to avoid signature-based detection.
— *Feature 6 and 7: the number of bytes allocated through string operations, and the number of likely shellcode strings* — these features are included to capture the characteristics of common attack techniques, such as heap overflow.
— *Feature 8: the number of instantiated components* — as we discussed in Section 2.2, attackers often attempt to exploit a number of vulnerabilities to maximize their success rate.
— *Feature 9 and 10: the values of attributes and parameters in method calls, and the sequence of method calls* — these features are included to encode the unique behavior of the vulnerability exploit phase.

All the signature features are fed into a machine learning algorithm proposed in Kruegel and Vigna [2003] to train an anomaly model. Then, the model is used to detect attacks by assigning a score to the webpage of interest. The higher the score, the higher the probability of its being anomalous (*i.e.,* triggering drive-by download attacks). 150,000 URLs were examined by JSAND. It has been shown that by selecting a proper training set, JSAND can achieve both lower false positives and lower false negatives than other signature-based [Nazario 2009] and VM-based [Provos et al. 2008] approaches.

Complementing the 10 signature features proposed in JSAND, Song et al. [2010] identifies anomaly communication signatures between different web browser components under drive-by download attack scenarios by modeling the common attack workflow (as discussed in Section 2.2). The authors demonstrated that such communication behavior could offer accurate and robust signatures for identifying real-world drive-by download attacks. Alter-

(1) Vulnerable PHP Code and XSS Attack Example

```
echo "Here is what you wrote: $_GET['input']"
```

```
http://example.com/?input=<script>alert();</script>
```

(2) Vulnerable PHP Code and SQLI Attack Example

```
$sql = "SELECT * FROM product WHERE pid = '".$_GET['pid']."'"
```

```
http://example.com/?id=1';DROP TABLE product;#
```

Fig. 8: Simple Vulnerable PHP Script and Attack Examples

natively, Zozzle [Curtsinger et al. 2010] explores the possibility of using JavaScript code signatures to detect web-based malware attacks. Zozzle uses a semi-automated machine-learning approach to extract a large set of code signatures from previously detected malicious JavaScript code. It then uses this signature database to do fast string matching with any new code observed. Zozzle can be implemented as a JavaScript engine extension, which detects malicious JavaScript code and the potential web-based malware attacks on-the-fly. Other early signature-based work includes Snort IDS [Roesch 1999], which performs data-flow signature matching of attack patterns at the network level, and PhoneyC [Nazario 2009], which performs similar techniques at the application level.

It is worthwhile to note that Toth and Kruegel [2002], Akritidis et al. [2005], Polychronakis et al. [2007] and Ratanaworabhan et al. [2009] investigate signature-based approaches for detecting the stack or heap overflow techniques. Such memory overflow techniques are widely used by attackers to inject malicious shellcode for conducting drive-by download attacks. These studies use emulation approaches at either the network or application level to detect memory overflow signatures. Various heuristics are proposed in these studies to improve the detection accuracy and performance overhead.

5. IDENTIFICATION OF APPLICATION VULNERABILITY

As we discussed in Section 2.3, unvalidated user input is one of the principle causes that allows attackers to use legitimate websites used by the attackers to propagate malware. Indeed, the Top Ten Project [2011] of the Open Web Application Security Project (OWASP) has listed injection attacks as the number one threat to web application security. One example of this injection attack is commonly known as the cross-site scripting (XSS) attack [Kirda et al. 2006; Jim et al. 2007; Wassermann and Su 2008]. In the case of XSS, the attackers leverage vulnerable functions of a web application to send malicious JavaScript code as the output of the web application to other users' browsers for execution. Another well-known problem is the SQL injection (SQLI) attack [Halfond et al. 2006; Geneiatakis et al. 2006]. In the case of SQLI, the attackers supply carefully-crafted input into vulnerable web applications, which leads to modifications or leakage of important information (*e.g.,* administrator credential) stored in the back-end database system. Figure 8 shows examples of vulnerable scripts and possible attacks.

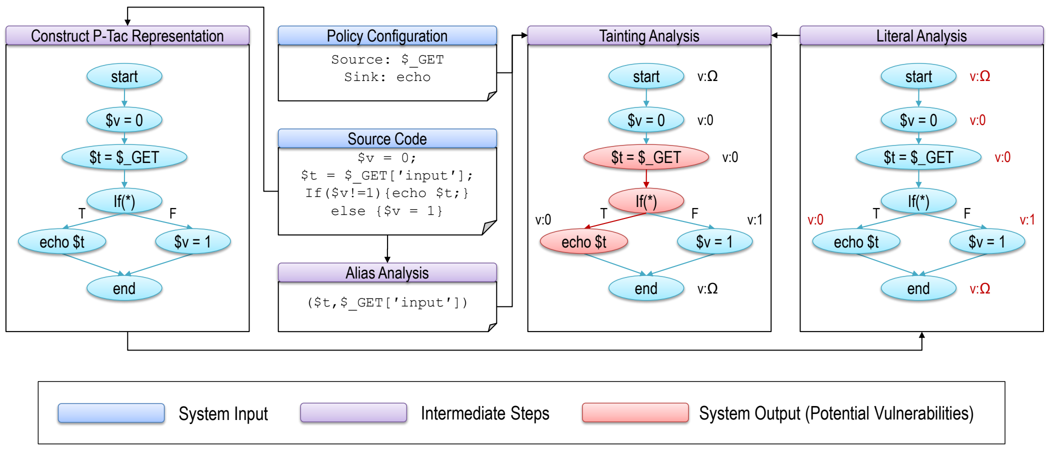
The main functionalities of a web application have long been implemented by its server-side components. Therefore, many of the previous research efforts have been devoted to finding server-side code vulnerabilities. We survey this research branch in Section 5.1. As the complexity of the client-side components keeps increasing, more recent attention has been given to study the potential flaws. This research branch is discussed in detail in Section 5.2. A brief summary of the surveyed work in this section is shown in Table VI.

Category	Paper	Contribution Summary
Vulnerability Identification of Server-side Code	Jovanovic et al. 2006a	Using static analysis of PHP code to identify XSS vulnerabilities.
	Huang et al. 2004; Xie and Aiken 2006; Wassermann and Su 2007; Lam et al. 2008	Improving existing static analysis techniques of the server-side code component by enhancing the analysis precision, scalability, or the policy specification process.
	Balzarotti et al. 2008	Combining the use of static and dynamic code analysis to improve the vulnerability identification capability.
	Bau et al. 2010; Balduzzi et al. 2011	Using black-box code testing techniques to identify server-side code vulnerabilities.
Vulnerability Identification of Client-side Code	Saxena et al. 2010 (FLAX); Saxena et al. 2010 (Kudzu)	Combining the use of tainting analysis and black-box fuzzing tests to identify client-side validation vulnerabilities.
	Chugh et al. 2009; Guarnieri and Livshits 2009; Chang et al. 2011	Using various static code analysis techniques to identify potential client-side code vulnerabilities.

Table VI: Summary of Vulnerability Identification Techniques

5.1. Server-side Code Vulnerability

A well-studied approach to discovering code vulnerabilities is tracking the information flow within web applications [Sabelfeld and Myers 2003]. Three important terms are defined within this context: (1) *source*: untrusted data or operations; (2) *sink*: critical data or operations that are vulnerable and need to be protected; and (3) *policy*: the specification of security requirements, for example, the confidentiality or the integrity of data. Using data tainting and flow analysis techniques, the web application code is analyzed to identify the information flow from sources to sinks, or vice versa. Depending on the policy, a flow from sources to sinks may violate the integrity requirement (*e.g.,* XSS attacks). At the same time, a flow from sinks to sources may violate the confidentiality requirement (*e.g.,* website cookies that are leaked to the attackers). Such violations are then caught and reported as vulnerabilities.



```
Source: $_GET
Sink: echo
```

```
$v = 0;
$t = $_GET['input'];
If($v!=1){echo $t;}
else {$v = 1}
```

```
($t,$_GET['input'])
```

Fig. 9: Example of Pixy System Usage

5.1.1. Static Code Analysis. In Jovanovic et al. [2006a], Pixy, the first open source tool for statically detecting XSS vulnerabilities in PHP code, was developed.[3] An example of the Pixy usage is illustrated in Figure 9. In Pixy, the sources include several types of program entry points, such as `GET`, `POST` and `COOKIE` arrays. All the routines that return data to the browser, such as `echo`, `print` and `printf` are considered to be the sensitive sinks. The notion of a *sanitation routine* is defined as the functions that can destroy potentially malicious characters contained in the sources. Several built-in PHP functions, such as `htmlentities` and `htmlspecialchars` are considered to be proper sanitation functions. The goal of Pixy is to identify the integrity vulnerabilities: any information flow from sources to sinks without sanitation is considered to be problematic.

Static data flow analysis is used in Pixy to achieve the design goal. The Pixy system takes PHP source code files as its input. First, parse trees are constructed by parsing the PHP input files. These parse trees are then transformed into an intermediate representation that is called *P-Tac*. The P-Tac representation is a form of the three-address code [Aho et al. 1986] enhanced with the control flow graph for every invoked function. The last and the most important step is to conduct a flow analysis over the intermediate P-Tac representation. This analysis identifies all of the sensitive sinks that can be reached by source data flows as potential vulnerabilities.

Several important features are implemented in Pixy to achieve better analysis accuracy. First, it performs an alias analysis to correctly propagate the taint value to both the variables and all of their aliases. This alias analysis technique is proposed in Jovanovic et al. [2006b] for reference-based scripting language. Second, a literal analysis is used to collect information about the literal values that variables or constants may hold at every program point. The literal information is then used by the flow analysis to evaluate branch conditions and to ignore program paths that cannot be executed at run-time. Third, all the built-in PHP functions except for the sanitation routines are thought to preserve data flow by default. The same is true for user defined functions, unless a specification of the true behavior is provided. This design choice is made to minimize the false negatives. False positives can often be solved by manual inspections; however, there is no effective way to handle false negatives. The authors used Pixy to scan six real-world PHP web applications and reported

[3]In this paper, we choose PHP as the representative server-side scripting language due to its popularity in practice. However, the techniques being surveyed in this paper can have much broader applicability.

about 50 vulnerabilities. The authors also found that Pixy suffered from both false positives and false negatives that were caused by certain dynamic properties of the PHP language, which the static analysis is incapable of handling.

Similar to Pixy, Huang et al. [2004], Xie and Aiken [2006], Wassermann and Su [2007] and Lam et al. [2008] focus on the static information flow analysis to detect the vulnerabilities of web applications. The differences among these studies varies, depending upon the analysis capabilities or target language. Besides identifying the vulnerabilities, Huang et al. [2004] also inserts runtime guard code to ease the vulnerability patching process. In Xie and Aiken [2006], the authors improve the capabilities of static code analysis to handle many dynamic features of the PHP language. Meanwhile, Wassermann and Su [2007] proposes a technique to automate the policy specification process and the scalability of existing code analysis systems. A declarative policy specification language is proposed in Lam et al. [2008], and the authors demonstrate several examples to show that the proposed policy language is expressive enough for encoding succinct descriptions of many real-world vulnerabilities.

5.1.2. Combining Static and Dynamic Analysis. False negatives could occur due to the assumption that if a sanitation function was applied on the information flow path from source to sink, then this path is safe. However, this assumption does not always hold. For example, Su and Wassermann [2006] discussed the possibility of subtle SQL injection vulnerabilities that can be exploited even when the input is processed by a built-in PHP sanitation routine. This issue is more problematic when custom checking routines are used. The correctness of these custom checking cannot be guaranteed due to the lack of systematic testing.

To address this problem, Balzarotti et al. [2008] extended Pixy with dynamic code analysis capabilities to design a new tool called Saner. The capabilities of Pixy are preserved to identify all the information flow from source to sink. By over-approximation of the values that each string variable can hold for every point in the program, Saner then checks whether these values contain any element that may pose a security risk. A dynamic analysis is then performed by executing the code with a large set of different inputs that includes many ways of encoding and hiding malicious characters. By doing this testing, Saner attempts to confirm the existence of vulnerabilities by finding program input that can bypass the sanitation routines and reach the sinks.

The dynamic testing phase of Saner is conducted in two steps. First, Saner constructs a sanitization graph for each source and sink pair provided by the static analysis. In the second step, a number of attack strings are used to confirm the potential vulnerabilities. If the testing process confirms that there is no effective sanitation between a source and a sink, it triggers an alert and provides the problematic program path along with sample attack strings. This information can then be leveraged by the developers to fix the vulnerability. The authors showed that Saner can identify many previously unknown vulnerabilities in real-world web applications.

5.1.3. Black-box Code Testing. Another type of approach, black-box code testing, is also widely used in practice to find software bugs. Instead of analyzing the program source code, black-box testing feeds known malicious input into web applications as test cases to identify potential vulnerabilities. In Bau et al. [2010], eight automated black-box testing tools from leading vendors are compared. The comparison shows that these tools are effective in identifying well-studied vulnerabilities, but they are insufficient for discovering more complex and subtle flaws.

Focusing on the `URL` parameter as the sensitive source, Balduzzi et al. [2011] proposes PAPAS — the first system that automatically discovers `URL` parameter pollution vulnerabilities of web applications. The PAPAS system adopts a black-box testing approach to inject `URL` parameter test cases into target web applications and uses a set of heuristics to determine potential vulnerabilities. By using the proposed techinque to analyze about 5,000 popular websites, the authors observed that about 30% of these sites contain vulnerablities that can be exploited to conduct various SQLI or XSS attacks.

5.2. Client-side Code Vulnerability

Similar to the server-side code, client-side components of web applications can also be vulnerable. Such client-side components are usually written in interpreted script language (*e.g.,* JavaScript) to enhance interactivity and to provide functionalities. Client-side script has the power to interact with webpages and the browser; however, it also opens doors for many browser-based attacks. Several studies have been conducted to enhance the current browser implementation of the same-origin policy [Jackson and Wang 2007; Crites et al. 2008]. Such work focuses on designing a secure interface for untrusted principals (*e.g.,* webpages from different domains) to communicate with each other, but the secure interfaces are not the solutions for addressing the emerging security challenges and vulnerabilities (*e.g.,* unsanitary user input, and insecure mashup paradigm) related to the web-based malware problem, as we discussed in Section 2.3.

5.2.1. The Status Quo of Insecure JavaScript Practices. A large-scale measurement study on insecure JavaScript practices is presented in Yue and Wang [2009]. In this study, two types of insecure practices were considered: (1) *JavaScript inclusion*: including JavaScript files from external domains into the top-level document of a webpage; and (2) *dynamic JavaScript generation*: using reflective functions, such as `eval` to generate new scripts at run-time. Both types of insecure practices can lead to vulnerabilities and allow for the injection of malicious scripts from the attackers.

As a part of Yue and Wang's study, the homepages of 6,805 popular websites in 15 different genres were visited. It has been observed that 66.4% of the measured websites have the insecure JavaScript inclusion problem. More than 74.9% of the measured websites use dynamic script generation techniques. In other words, insecure JavaScript practices are prevalent on the Web. These results warn both website developers and administrators to pay serious attention to these issues and to use safe alternatives to avoid them.

5.2.2. Discovering Client-side Code Vulnerability. In Saxena et al. [2010], a class of validation vulnerabilities is formally analyzed for client-side code. This client-side validation (CSV) vulnerability is defined as a programming flaw that results from using untrusted source data in a critical sink without sufficient validation. Data from the external web principal or end users are defined as untrusted sources. In fact, the sources could be messages from remote servers, cross-window communications (*e.g.,* HTML-5 `postMessage`) or user input (*e.g.,* form fields on a webpage). The sinks are defined to include various critical data structures (*e.g.,* `document.cookie` and `document.forms[*].action`), functions (*e.g.,* `eval` and `document.write`), which might be vulnerable to attacks (*e.g.,* code injection, parameter injection). Similar to Saner, the authors propose a code analysis technique named FLAX, which employs taint-enhanced black-box testing.

FLAX is inspired by Ganesh et al. [2009], and it uses a two-step process to identify CSV vulnerabilities. This two-step process is designed to reduce program size and the corresponding search space for the black-box test. In the first step, the web application is executed with the initial input I to conduct a dynamic taint analysis at the character level. The taint analysis identifies all the potential uses of untrusted source data in critical sinks. It then extracts an input set I_S, on which the arguments of a sink operation S depend. All the program statements that are directly dependent on I_S are then extracted into an executable stand-alone sub-program, termed an *acceptor slice*. In the second step, each extracted acceptor slice is fed with test cases to find vulnerabilities. This process is sink-aware as the test input is specifically designed for different types of sinks. For example, a large corpus of XSS attack vectors is used to test the sinks that are vulnerable to code injection attacks. The system design of FLAX is illustrated in Figure 10.

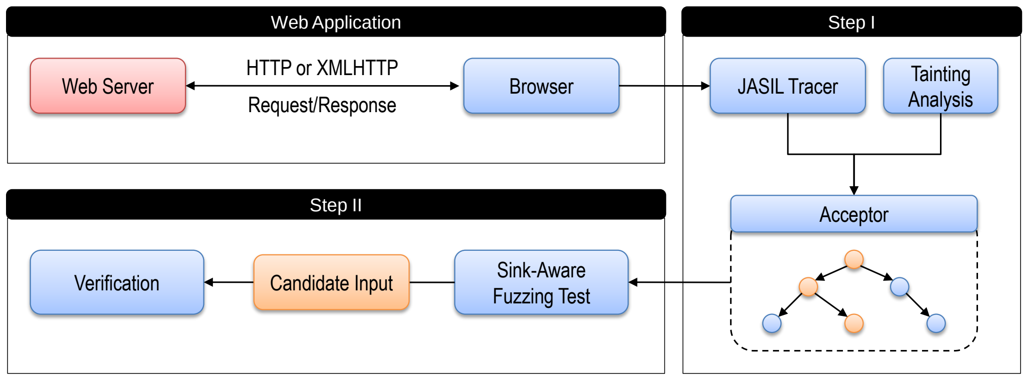
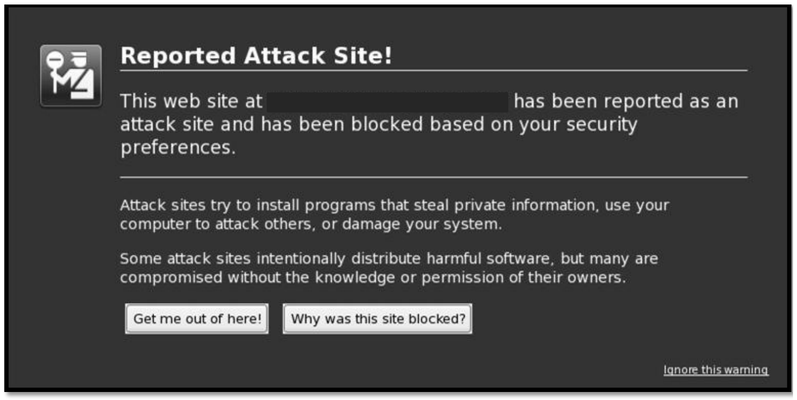


Fig. 10: FLAX System Design

Important design choices were made to handle various technical challenges of implementing FLAX. First, a simplified version of JavaScript named JASIL has been designed to facilitate the analysis process. Second, when tainted data is sent to external servers, FLAX performs the longest common substring matching over the returned data. Finally, implicit sinks are also considered to reduce false negatives. By running FLAX over 40 web applications, 11 previously unknown vulnerabilities were discovered. After manual verification of the application code, the authors argue that FLAX has a low false positive and a low false negative rate. Using these identified vulnerabilities, four proof-of-concept attack scenarios were illustrated, including origin mis-attribution, code injection, application command injection and cookie corruption.

In Saxena et al. [2010], the authors enhance the design of FLAX and propose a new system called Kudzu for identifying CSV vulnerabilities. Test cases are supplied as input to the FLAX system for conducting blackbox fuzzing test. Unlike FLAX, Kudzu generates high-quality test cases automatically. Further, Kudzu uses a new language to express string constraints and adopts a novel constraint solver to explore the symbolic execution space of JavaScript code. By identifying code injection vulnerabilities in real-world applications, Kudzu demonstrates its effectiveness without suffering from false positives.

Unlike FLAX, Chugh et al. [2009], Guarnieri and Livshits [2009] and Chang et al. [2011] adopt different static code analysis techniques to identify insecure JavaScript program behavior that leads to potential client-side code vulnerabilities. GateKeeper [Guarnieri and Livshits 2009] is a code validation tool that is useful for web widget portals (*e.g.,* iGoogle) to enforce customized security policies before allowing new widgets to be published. By disallowing dynamic code generation features of the JavaScript language, GateKeeper uses mostly static code analysis (*e.g.,* pointer analysis, static call graph) to enforce security policies. The satisfaction of the corresponding policies by a widget is considered as a certification process of the widget portal. Chugh et al. [2009] is a staged information flow approach to enforce data-flow policies for JavaScript code. The proposed scheme identifies syntactically enforceable rules during an off-line analysis stage and enforces them during the execution of JavaScript code to achieve minimal performance overhead. ToMaTo [Chang et al. 2011] is a development tool that combines a novel trust policy language and a static code analysis engine to examine vulnerabilities introduced due to the usage of external JavaScript libraries in mashup web applications. ToMaTo enables the mashup developers with three essential capabilities for identifying potential vulnerabilities to build trustworthy JavaScript code mashups: trust policy specification, policy adherence evaluation, and vulnerability (*i.e.,*



Reported Attack Site!

This web site at [redacted] has been reported as an attack site and has been blocked based on your security preferences.

Attack sites try to install programs that steal private information, use your computer to attack others, or damage your system.

Some attack sites intentionally distribute harmful software, but many are compromised without the knowledge or permission of their owners.

Fig. 11: Screen-shot of Google Safe Browsing

policy violation) handling. Overall, these static analysis approaches are sound but suffer from false positives due to the nature of the techniques.

6. PROTECTION AGAINST WEB-BASED MALWARE

The last line of defense against web-based malware is to protect the end users from being infected. As we saw in Section 2.4.1, instead of focusing specifically on web-based malware, traditional protection mechanisms, such as anti-virus software, sandbox techniques and security patches, are generally effective against a larger range of threats. Notably, alternative patching schemes have been explored in Wang et al. [2004] and Reis et al. [2006]. Their studies provide techniques for patching known vulnerabilities of client machines at network level to ease the patch application process. Despite the usability advantage, the accurate specification of network-level patches for the corresponding host vulnerabilities is still a challenging task.

The detection techniques discussed in Section 4 can often be extended to act as protection mechanisms as well. The idea is that if malice can be effectively detected, and the end users are notified about the risk, then they can avoid visiting those websites that expose them to danger. For instance, the VM-based detection technique adopted in Provos et al. [2008] has been used in the Google Safe Browsing Project [2011], which constructs a blacklist of detected landing sites. Browsers, such as Firefox and Chrome, which incorporate the blacklist, can dissuade end users from visiting high risk websites (see Figure 11). Moreover, the detection mechanisms can also be encapsulated as proxy services, which perform on-line detection of attacks before sending unsafe content to end users [Moshchuk et al. 2007; Rieck et al. 2010].

In this section, we discuss two categories of approaches: (1) sandbox systems, which are specially designed to prevent web-based malware infection; and (2) URL reputation schemes, which improve the dynamics and the predictability of static malware blacklists. A brief summary of surveyed work in this section is shown in Table VII.

6.1. Building Attack-Agnostic Sandbox

Sandboxing is a well-studied and widely-used security mechanism to execute untrusted code by restricting its privileges in a trusted system. In Lu et al. [2010], the authors propose a smart sandbox mechanism, called BLADE, to protect end users from all forms of drive-by download malware attacks. In BLADE, the end user is the root of trust. All downloaded content with explicit user consent is considered trustworthy. It is important to note that BLADE will not work for web-based malware delivered through social engineering tricks.

Category	Paper	Contribution Summary
Attack-Agnostic Sandbox	Lu et al. 2010	Using sandbox mechanism to isolate untrustworthy downloads created by web-browsers.
	Cox et al. 2006; Li et al. 2011	Providing strong isolations between the execution of untrustworthy web applications and the host operating system.
	Jain et al. 2008	Providing a sandbox system that both limits the effects of attacks and simplifies the post-intrusion recovery process.
	Grier et al. 2008; Reis and Gribble 2009;	Using sandbox mechanism for secure web browser designs.
URL Reputation	Ma et al. 2009	Building a dynamic and predictive blacklist using `URL` reputation.
	Antonakakis et al. 2010	Building a dynamic reputation system for domain names.

Table VII: Summary of Web-based Malware Protection Techniques

The goal of BLADE is to put content downloaded without a user's consent into an isolated sandbox environment to prevent any damages it might cause. The authors argue that BLADE is attack-agnostic, as it does not require any understanding of the exploit techniques or vulnerabilities. In contrast, it needs to correctly capture user's consent.

The system design of BLADE is shown in Figure 12. By default, BLADE puts all files downloaded from the browser process into a non-executable secure zone. Once BLADE can correlate a downloaded file with a user's permission, the file will be moved out of the sandbox. Therefore, the key challenge of BLADE is to build a robust, user-transparent, and browser-independent correlation mechanism between the user authorization and the downloaded content. BLADE achieves this goal in four steps: (1) BLADE monitors kernel window events to look for the appearance of a download consent dialog triggered by the browser. A handful of user interface (UI) signatures are used to identify these consent dialog windows and to extract (`URL`, file path) information from them. (2) Once such a dialog is discovered, BLADE begins to sense user-invoked hardware signals (*e.g.,* mouse clicks or keystrokes), which may indicate the user's consent. Once such a signal has been observed, the correlation process starts. (3) The correlation process establishes proper mappings between user download authorizations and downloaded files. The (`URL`, file path) pair extracted in step 1 is used to provide information for the correlation process. File path information is used as the ID to find the candidate file from the sandbox zone. Meanwhile, the `URL` is used to double check against a log of TCP sessions to increase the robustness of the

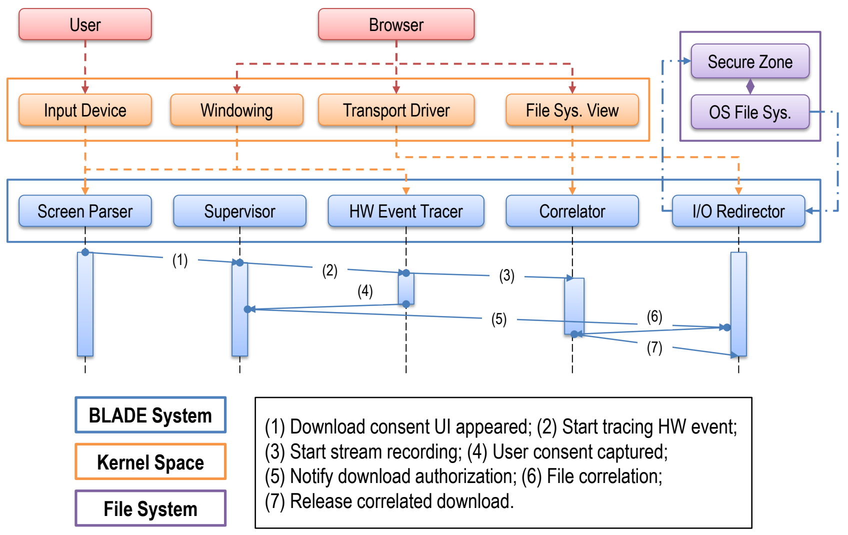

User
Browser
Secure Zone
OS File Sys.
Input Device
Windowing
Transport Driver
File Sys. View
Screen Parser
Supervisor
HW Event Tracer
Correlator
I/O Redirector
(1)
(2)
(3)
(4)
(5)
(6)
(7)
BLADE System
Kernel Space
File System
(1) Download consent UI appeared; (2) Start tracing HW event;
(3) Start stream recording; (4) User consent captured;
(5) Notify download authorization; (6) File correlation;
(7) Release correlated download.

Fig. 12: BLADE System Design

correlation result. (4) Once a correlation is established, the downloaded content with a user's permission will be redirected from the sandbox to the OS file system. Security policies are enforced by BLADE to enable a complete containment of disk footprints affected by the downloaded content. The authors evaluate BLADE against 4,000 malicious webpages that contain drive-by download exploits. The experiment showed that BLADE is effective against all these attacks and only introduces a time delay of 4% compared with benign content downloading.

Cox et al. [2006] and Li et al. [2011] propose alternative sandbox schemes for isolating the execution of untrustworthy web applications. Cox et al. [2006] propose a new trusted software layer, the Tahoma Web browsing system, to provide strong isolations between different web applications and the host operating system. Having the same security management capabilities of native applications on current operating systems, web applications are explicitly controlled and managed by the Tahoma system. By utilizing a carefully designed virtual machine system, the Tahoma system achieves strong isolation and safety without compromising the application performance. WebShield [Li et al. 2011] is a sandbox system that implements the principle of program isolation at the network level. The WebShield system treats all the JavaScript code of web applications as untrustworthy and executes it inside a sandbox. Serving as a proxy server, WebShield only forwards the consequences of JavaScript code execution (*e.g.,* DOM modifications) to the client-side browsers for rendering. The interactivity of web applications is also preserved by implementing special event handlers and communication protocol between the client-side browser and the sandbox proxy. The authors argue that WebShield has more general capabilities than existing network-level isolation schemes such as Reis et al. [2006] and Moshchuk et al. [2007]. Solitude is a sandbox system that is designed to both limit the effects of attacks and simplify the post-intrusion recovery process [Jain et al. 2008]. Instead of focusing on any specific type of attack, Solitude provides a transparent and isolated environment for running all types of untrusted applications. Moreover, Solitude supports a more fine-grain privilege model than BLADE.

The Chromium web browser also provides a sandbox mechanism to isolate different web programs by taking advantage of existing OS-level isolation between different processes

[Reis and Gribble 2009]. However, it is designed to enhance the current same-origin-policy implementation to defend against XSS attacks. Indeed, malware delivered through client-side exploits against Chrome has already been observed [Chrome Malware 2010]. Similarly, in Grier et al. [2008] the authors also address the principle of secure web browser design. Grier et al. [2008] advocates a scheme that uses formal methods to drive the application of the security design principles of modern operating systems. The overall idea is to design a web browser that consists of function modules and a small browser kernel, where the browser kernel controls and interposes communications between all the subsystems and components. The authors argue that the proposed scheme will be less error-prone and ease the task of pinpointing attack scenarios and potential vulnerabilities.

6.2. Calculating URL Reputation

Current ways to build a URL blacklist depend on explicit feedback; malice is observed by either manual inspections or automatic detections. Inevitably, these blacklists are not complete — many malicious URLs are not blacklisted. There are two main reasons for this defect: (1) the URL is "new", meaning that it has not yet been evaluated; or (2) the evaluation result is incorrect due to the false negative exhibited by the detection approach.

Motivated by the insufficiency of a static blacklist, Ma et al. [2009] proposed an approach to build a dynamic and predictive blacklist using URL reputation. Note that this technique is not limited to detecting web-based malware problems. It is also effective for other web threats, such as phishing and spamming, as demonstrated by Ma et al. [2009]. The proposed reputation scheme assigns a binary reputation value to the URL, indicating whether it is malicious or not. In Ma et al. [2009], various features and classification models were studied to gain a better understanding of the problem.

Instead of analyzing the content pointed at by the URL, two sets of light-weight features are considered when building the URL reputation. The first is a lexical feature set, which is the textual properties of the URL itself. Lexical features include the length of the hostname, the length of the entire URL, the number of dots in the URL and a "bag-of-words" representation of tokens (a string delimited by '.', '?', '/', '=', '-', etc.) in the URL. The second feature set, the host-based feature set, contains information about the host of a website. IP address properties (*e.g.,* AS and prefix information, whether the IP address appears in any other blacklists), WHOIS properties (*e.g.,* domain registration data, registrar information), DNS properties (*e.g.,* time-to-live value), and geographic properties (*e.g.,* the country to which the host IP belongs, the speed of the uplink connection) are included as the host-based features. A feature comparison study showed that the following four features are the most relevant for correct classifications: the number of dots in a URL, whether the hostname contains an IP address, the WHOIS registration date, and the membership in six static blacklists.

Three basic machine learning models were studied by the authors: the naive Bayes (NB), the support vector machine (SVM), and the logistic regression (LR). A comparative study between the three models shows that SVM and LR produce similar results, and both outperform NB model. Moreover, SVM has smaller training and testing overhead, making it more desirable when the performance is a concern. In contrast, LR often needs more computation power to train, but the resulting model is easier to interpret in terms of relevant and irrelevant features. 30,000 benign URLs and 10,000 malicious URLs were collected as the data set to validate the reputation scheme. The data set is grouped into four subsets, and 95% predication accuracy was observed for the training and testing reputation model within each subset. However, the accuracy decreased significantly when the training data and testing data are from different subsets. The authors argue that a comprehensive and representative training dataset is critical to make the proposed reputation scheme effective in the real world. How to obtain such a training dataset is still an open challenge.

Comparator	**IV**	**DA**	**PA**
Deployment	Global web application deployment	Partial, third-party deployment	Global user system deployment
Effectiveness	Both false positives and false negatives	Both false positives and false negatives	Mainly false negatives
Autonomy	High human involvement	Limited human involvement	Almost no human involvement
Adaptiveness & Gamesmanship	Attack-dependent	Attack-dependent	Attack-independent

Table VIII: Approach Comparison Summary (IV: Identification of Web Application Vulnerability; DA: Detection of Web-based Malware Attacks; PA: Protection against Web-based Malware Attacks)

There is little related work in the literature focusing on building dynamic reputations for web-based malware distribution sites. Instead of focusing on the URL, Notos [Antonakakis et al. 2010] is a dynamic reputation system for domain names. Notos uses reputation to quantify the probabilities of domain names being used to participate in botnet or spamming activities by adopting a hierarchical machine learning model. Notos does not use any lexical features to calculate reputation, but focuses only network features. With the same design goal as Notos, EXPOSURE [Bilge et al. 2011] is another system for identifying malicous domains used in cyber attacks. Adopting a different feature set and learning algorithm than Notos, the authors argue that the EXPOSURE system achieves better accuracy and requires less training data for building an accurate classifier. However, it is still unclear how to apply the schemes proposed by Notos and EXPOSURE for the effective classification of domain names used by malware landing or distribution sites.

7. DISCUSSION

In the previous sections, we introduced different techniques for defending against web-based malware. Now, we examine these methods comparatively. Our dimensions for comparison are not intended to be exhaustive. Instead, we choose attributes which highlight the strengths and weaknesses of each approach and help to design a comprehensive defense strategy. Table VIII summarizes the comparison of the surveyed approaches.

7.1. Deployment Scale

To defend against a rising security threat is a complicated issue in the real world. Even if one makes the assumption that the proposed approaches can work perfectly, there are still many other factors that can reduce the desired impact in practice. One aspect is the deployment scale: how widely one approach needs to be deployed in order to achieve the global defense goal.

Ideally, every web application on the Internet should be properly checked for potential vulnerabilities, in order to minimize the risk of becoming a landing website. However, there are two important factors that can be taken into consideration to relax this requirement. First, a small percentage of top websites attracts a large percentage of the global web traffic. These popular websites should be given the highest priority to check for vulnerabilities. If these top sites could always keep themselves safe, their users would also be safe. Secondly, many web applications are built upon popular web application platforms. For example, many on-line forums are powered by open source web applications, such as Phorum and phpBB. These platforms serve as the "template" with customization capabilities to produce new applications. Since the code of these platform is highly reused, it is important to make sure that they contain as few vulnerabilities as possible. The same is true for client-side code, as many web applications are dependent on a handful of popular JavaScript code libraries, such as jQuery and YUI. Moreover, it is reasonable to assume that both top website administrators and popular web platform developers have more resources and better expertise for this task. As a result, it is also their responsibility to contribute more towards building a safe web ecosystem.

Attack detection mechanisms can be deployed on a much smaller scale and remain effective. Indeed, Google checks a large portion of existing webpages continuously in its Safe Browsing project, with only local deployments [Provos et al. 2008]. If we can reduce the performance cost of detection techniques and add more computing resources, it is still possible to expect real-time and rather complete detection of attacks in the wild. Further, detection systems can also focus more on the top popular websites to be more cost-effective. However, the impact of a successful detection mechanism would be limited without reaching out to the end users.

On the other hand, the protection mechanisms surveyed in this paper require a complete coverage of all end hosts to achieve the best results. From the attacker's perspective, there is almost no difference between different victim machines, *i.e.,* "every machine counts". However, looking at the history of defending against other computer security threats, it is reasonable to argue that achieving a full deployment is not likely to happen in reality. Therefore, designing better techniques to ease the deployment process and to achieve a better coverage percentage might be a very desirable next step.

7.2. Comparative Effectiveness

Another comparison can be made by assuming a full deployment of all proposed approaches and focusing on their comparative effectiveness. Given the different nature of proposed approaches, we want to understand their mitigation limitations.

Vulnerability identification mechanisms cannot identify all potential vulnerabilities. The code analysis techniques often try to identify the necessary conditions of successful exploits to minimize false negatives by increasing false positives. However, without the formal specifications of correct program behavior, many alerts triggered by the vulnerability scanners might simply be ignored by developers to preserve functionality. On the other hand, black-box testing only uses known attacks as test cases. Presumably, this testing process will suffer from both false positives and false negatives. Further, the code of web applications is often quickly evolving to meet new user requirements. The vulnerability checking process needs to be performed every time the code changes. Otherwise, new vulnerabilities introduced by the code modification cannot be detected. In other words, none of the existing vulnerability identification mechanisms can remove all the bugs and flaws from web applications. The risk of buggy software will always be present.

As discussed in Section 4.2, VM-based detection mechanisms suffer from false negatives given the complexity of configuring the most vulnerable honeypot to detect all attacks. Meanwhile, signature-based detection mechanisms use either pattern matching or machine learning techniques for detection. Both methods depend on detected attacks to build a signature database or an anomaly model. However, there are still no known ways to build a complete or at least a representative attack vector set. Moreover, both the attack signature and the anomaly model often capture the necessary conditions of successful attacks. Therefore, signature-based detection mechanisms will have both false positives and false negatives. It is highly possible that new attack types cannot be detected by these approaches until a large scale outbreak is observed.

A URL reputation mechanism has the same types of problems as signature-based detection. In contrast, the sandbox technique seems to be very promising, if it can be fully deployed. The only drawback is that it assumes that end users are the root of trust. However, end users may often make incorrect security decisions due to their lack of necessary knowledge or their victimization by social engineering tricks. Regardless, for attacks that are fully automated (*e.g.,* drive-by download), the smart sandbox system forms very effective protection.

7.3. Degree of Autonomy

We next examine the degree of autonomy from which each of the proposed approaches operates. That is, what role do humans play in the operation of these approaches? This is an important issue to understand since humans can sometimes be the most unreliable link of the entire defense chain.

Vulnerability identification mechanisms need a great deal of human participation. The alerts triggered by the scanning tools need to be examined by the developers one by one to decide whether true vulnerabilities are present. Sometimes, using the scanning tools requires deep knowledge of potential attacks. Further, developers are required to develop correct patches for the identified vulnerabilities without introducing new flaws.

Signature-based detection mechanisms also require humans to determine the correctness of detected attacks with further investigations. VM-based detection requires less human input. However, when it is used for detecting zero-day vulnerabilities, it still needs researchers to study the details in order to have a good understanding of the problem.

Two protection mechanisms surveyed in this paper are almost fully automated and require almost no human participation. A static or dynamic URL blacklist can be embedded in the browser itself and be shipped to end users. For instance, the Firefox browser implements the Google Safe Browsing API and blocks any visits to the malicious sites automatically. The BLADE sandbox systems are also designed to be transparent to end users once installed on the system. It extracts information from normal browsing behavior and uses it to reason about user authorizations without the need for extra input.

7.4. Adaptiveness and Gamesmanship

Finally, we discuss how proposed approaches need to be modified to adapt to new attacks, and how existing attacks might change themselves to evade these approaches.

To detect new types of vulnerability, the vulnerability identification mechanisms based on code analysis need to take new sources or sinks into considerations, or change the checking policies to include new security violations. However, some vulnerabilities will require significant improvement of the code analysis capability, *e.g.,* to support more dynamic language features. Black-box testing requires adding new attacks into their test cases to be adaptive. To evade these mechanisms, the attackers need to use more advanced and subtle techniques as demonstrated in Bau et al. [2010].

If new attacks are designed to target new vulnerabilities, the VM-based detection mechanism needs to include these vulnerabilities into the VM image to still be effective. Otherwise, no modifications are needed. For the signature-based detection mechanism, it requires including new attacks in the training dataset to learn special characteristics from these attack instances. To evade both of the detection mechanisms, malicious landing sites can perform a reverse Turing test (*e.g.,* CAPTCHA) to judge whether the end user is a human or a machine. Then they can deliver only benign content, if no human user is involved. Moreover, if the attackers have knowledge of the detection infrastructure, for example, IP addresses, then the landing sites can simply block detections by using an IP blacklist.

The URL reputation calculation also needs to include new malicious links to be adaptive, similar to the signature-based detection mechanism. In contrast, the sandbox approach is attack-agnostic, and it requires almost no changes for new attacks. Also, it is difficult to game the sandbox system under reasonable trust assumptions [Lu et al. 2010].

8. CONCLUSION

In this paper, we survey three categories of approaches to analyze, identify, and defend against the web-based malware problem. Each category of approach has its relative advantages and disadvantages, and each has been demonstrated to be effective via empirical evaluations. Further, we discuss how these approaches complement each other and how they can work together to form a comprehensive solution space. In order to advance the state-of-the-art, we have identified five promising directions for future research:

(1) *Building Benchmark Platforms*: Almost all the approaches suffer from either false positives or false negatives; however, there is no commonly accepted data set or testing framework to comparatively evaluate their effectiveness. Therefore, a well designed benchmark framework is clearly needed to scientifically study and compare different proposed approaches.
(2) *Securing Code Mashup*: The client-side code of web applications can be reused and dynamically loaded from external sources. This code mashup requires a different security model than any traditional programming paradigms. Given the prevalence of client-side code mashups, it is imperative to design a sound approach to enhance the flexibility of the current mashup programming practice with guaranteed security.
(3) *Studying Social Engineering Tricks*: Current detection approaches mainly focus on the web-based malware delivered through drive-by download attacks. The studies on malware delivered through social engineering tricks is very limited. However, as the technologies for mitigating drive-by download attacks become more mature and more broadly deployed, it is reasonable to assume that the attackers will focus more on using social engineering tricks to improve their chance of success.
(4) *Studying the Economy*: Understanding the incentives and the economic chains of the web-based malware is still an open area. Similar studies have been done for other network security problems such as e-mail spam [Kanich et al. 2008; Motoyama et al. 2010]. This type of study is critically important for security researchers to understand the landscape better and to design more effective defense strategies.
(5) *Studying the Epidemiology*: Existing detection mechanisms can be used to build the topology of the malware distribution infrastructure. However, there is no study on the liveness property of this topology: understanding how the connections between landing sites and distribution sites evolve over time. Similar studies have been done for traditional push-based malware propagation in Garetto et al. [2003]. An accurate epidemic model is therefore useful to evaluate how fast and prevalent a defense mechanism needs to be deployed to effectively fight against a web-based malware outbreak.

Given the scale and complexity of the web-based malware problem space, building a safe web browsing environment is the responsibility of every Web participant. On the whole, for a healthy and secure Internet ecosystem, a challenging road is still ahead for everyone in the foreseeable future.

REFERENCES

Aho, A. V., Sethi, R., and Ullman, J. D. 1986. *Compilers: principles, techniques, and tools.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Akritidis, P., Markatos, E., Polychronakis, M., and Anagnostakis, K. 2005. Stride: Polymorphic sled detection through instruction sequence analysis. In *Security and Privacy in the Age of Ubiquitous Computing*, R. Sasaki, S. Qing, E. Okamoto, and H. Yoshiura, Eds. IFIP Advances in Information and Communication Technology Series, vol. 181. Springer Boston, 375–391.

Antonakakis, M., Perdisci, R., Dagon, D., Lee, W., and Feamster, N. 2010. Building a dynamic reputation system for dns. In *Proceedings of 19th USENIX Security Symposium on USENIX Security Symposium.*

BAI, Y. AND KOBAYASHI, H. 2003. Intrusion detection systems: technology and development. In *17th International Conference on Advanced Information Networking and Applications, 2003. AINA 2003.* 710 – 715.

BAILEY, M., COOKE, E., JAHANIAN, F., XU, Y., AND KARIR, M. 2009. A survey of botnet technology and defenses. In *Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications Technology.* 299 –304.

BAILEY, M., OBERHEIDE, J., ANDERSEN, J., MAO, Z. M., JAHANIAN, F., AND NAZARIO, J. 2007. Automated classification and analysis of internet malware. In *Proceedings of the 10th international conference on Recent advances in intrusion detection.* RAID'07. Springer-Verlag, Berlin, Heidelberg, 178–197.

BALDUZZI, M., GIMENEZ, C. T., BALZAROTTI, D., AND KIRDA, E. 2011. Automated discovery of parameter pollution vulnerabilities in web applications. In *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS).*

BALZAROTTI, D., COVA, M., FELMETSGER, V., JOVANOVIC, N., KIRDA, E., KRUEGEL, C., AND VIGNA, G. 2008. Saner: Composing static and dynamic analysis to validate sanitization in web applications. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy.* IEEE Computer Society, Washington, DC, USA, 387–401.

BAU, J., BURSZTEIN, E., GUPTA, D., AND MITCHELL, J. 2010. State of the art: Automated black-box web application vulnerability testing. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy.* SP '10. IEEE Computer Society, Washington, DC, USA, 332–345.

BECK, D., VO, B., AND VERBOWSKI, C. 2005. Detecting stealth software with strider ghostbuster. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks.* DSN '05. IEEE Computer Society, Washington, DC, USA, 368–377.

BILGE, L., KIRDA, E., KRUEGEL, C., AND BALDUZZI, M. 2011. Exposure: Finding malicious domains using passive dns analysis. In *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS).*

BIND vulnerabilities 1998. Multiple Vulnerabilities in BIND. ftp://info.cert.org/pub/cert_advisories/CA-98.05.bind_problems.

BINSALLEEH, H., ORMEROD, T., BOUKHTOUTA, A., SINHA, P., YOUSSEF, A., DEBBABI, M., AND WANG, L. 2010. On the analysis of the zeus botnet crimeware toolkit. In *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on.* 31 –38.

CHANG, J., VENKATASUBRAMANIAN, K., WEST, A. G., KANNAN, S., SOKOLSKY, O., KIM, M. J., AND LEE, I. 2011. Tomato: A trustworthy code mashup development tool. In *5th International Workshop on Web APIs and Service Mashups, 2011. MASHUPS '11.*

CHRISTODORESCU, M., JHA, S., MAUGHAN, D., SONG, D., AND WANG, C. 2007. *Malware Detection.* Springer.

Chrome Malware 2010. New Drive-by Attack Targets Google Chrome Users . http://downloadsquad.switched.com/2010/04/20/new-drive-by-attack-targets-google-chrome-users/.

CHUGH, R., MEISTER, J. A., JHALA, R., AND LERNER, S. 2009. Staged information flow for javascript. In *Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation.* PLDI '09. ACM, New York, NY, USA, 50–62.

COVA, M., KRUEGEL, C., AND VIGNA, G. 2010. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th international conference on World wide web.* WWW '10. ACM, New York, NY, USA, 281–290.

COVA, M., LEITA, C., THONNARD, O., KEROMYTIS, A., AND DACIER, M. 2010. An analysis of rogue av campaigns. In *Recent Advances in Intrusion Detection*, S. Jha, R. Sommer, and C. Kreibich, Eds. Lecture Notes in Computer Science Series, vol. 6307. Springer Berlin / Heidelberg, 442–463.

COX, R. S., GRIBBLE, S. D., LEVY, H. M., AND HANSEN, J. G. 2006. A safety-oriented platform for web applications. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy.* IEEE Computer Society, Washington, DC, USA, 350–364.

CRITES, S., HSU, F., AND CHEN, H. 2008. Omash: enabling secure web mashups via object abstractions. In *Proceedings of the 15th ACM conference on Computer and communications security.* CCS '08. ACM, New York, NY, USA, 99–108.

CURTSINGER, C., LIVSHITS, B., ZORN, B., AND SEIFERT, C. 2010. Zozzle: Low-overhead mostly static javascript malware detection. Tech. Rep. MSR-TR-2010-156, Microsoft Research.

DANIEL, M., HONOROFF, J., AND MILLER, C. 2008. Engineering heap overflow exploits with javascript. In *Proceedings of the 2nd conference on USENIX Workshop on offensive technologies.* USENIX Association, Berkeley, CA, USA, 1:1–1:6.

Dasient Report 2010. Dasient Q3 Malware Update: Web-Based Malware Infections Double Since Last Year, Malvertising Attacks Continue Over Summer. http://blog.dasient.com/2010/11/normal.html.

FEILY, M., SHAHRESTANI, A., AND RAMADASS, S. 2009. A survey of botnet and botnet detection. In *Third International Conference on Emerging Security Information, Systems and Technologies, 2009. SECURWARE '09.* 268 –273.

FEINSTEIN, B. AND PECK, D. 2007. Caffeine monkey: Automated collection, detection and analysis of malicious javascript. In *Proceedings of BlackHat USA*.

FLEIZACH, C., LILJENSTAM, M., JOHANSSON, P., VOELKER, G. M., AND MEHES, A. 2007. Can you infect me now?: malware propagation in mobile phone networks. In *Proceedings of the 2007 ACM workshop on Recurring malcode*. WORM '07. ACM, New York, NY, USA, 61–68.

GANESH, V., LEEK, T., AND RINARD, M. 2009. Taint-based directed whitebox fuzzing. In *Proceedings of the 31st International Conference on Software Engineering*. ICSE '09. IEEE Computer Society, Washington, DC, USA, 474–484.

GARETTO, M., GONG, W., AND TOWSLEY, D. 2003. Modeling malware spreading dynamics. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*. Vol. 3. 1869 – 1879 vol.3.

GARFINKEL, S. AND SPAFFORD, G. 2001. *Web Security, Privacy and Commerce* 2nd Ed. O'Reilly & Associates, Inc., Sebastopol, CA, USA.

GENEIATAKIS, D., DAGIUKLAS, T., KAMBOURAKIS, G., LAMBRINOUDAKIS, C., GRITZALIS, S., EHLERT, S., AND SISALEM, D. 2006. Survey of security vulnerabilities in session initiation protocol. *IEEE Communications Surveys and Tutorials 8*, 68–81.

Google Safe Browsing Project 2011. Google Safe Browsing API Homepage. http://code.google.com/apis/safebrowsing/.

Google Web Index 2008. We Knew the Web was Big. http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html.

GRIER, C., TANG, S., AND KING, S. T. 2008. Secure web browsing with the op web browser. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*. IEEE Computer Society, Washington, DC, USA, 402–416.

GUARNIERI, S. AND LIVSHITS, B. 2009. Gatekeeper: mostly static enforcement of security and reliability policies for javascript code. In *Proceedings of the 18th conference on USENIX security symposium*. SSYM'09. USENIX Association, Berkeley, CA, USA, 151–168.

GULLI, A. AND SIGNORINI, A. 2005. The indexable web is more than 11.5 billion pages. In *Special interest tracks and posters of the 14th international conference on World Wide Web*. WWW '05. ACM, New York, NY, USA, 902–903.

HALFOND, W. G., VIEGAS, J., AND ORSO, A. 2006. A classification of SQL-Injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*. Arlington, VA, USA.

HOSSEINPOUR, F., BAKAR, K., HARDOROUDI, A., AND KAZAZI, N. 2010. Survey on artificial immune system as a bio-inspired technique for anomaly based intrusion detection systems. In *Intelligent Networking and Collaborative Systems (INCOS), 2010 2nd International Conference on*. 323 –324.

HUANG, Y.-W., YU, F., HANG, C., TSAI, C.-H., LEE, D.-T., AND KUO, S.-Y. 2004. Securing web application code by static analysis and runtime protection. In *Proceedings of the 13th international conference on World Wide Web*. WWW '04. ACM, New York, NY, USA, 40–52.

IDIKA, N. AND MATHUR, A. P. 2010. A survey of malware detection techniques. Tech. Rep. Tech Report 286, Purdue University, Department of Computer Science.

Internet Stats 2010. Internet World Stats. http://www.internetworldstats.com/stats.htm.

JACKSON, C. AND WANG, H. J. 2007. Subspace: secure cross-domain communication for web mashups. In *Proceedings of the 16th international conference on World Wide Web*. WWW '07. ACM, New York, NY, USA, 611–620.

JACOB, G., DEBAR, H., AND FILIOL, E. 2008. Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in Computer Virology 4*, 251–266. 10.1007/s11416-008-0086-0.

JAIN, S., SHAFIQUE, F., DJERIC, V., AND GOEL, A. 2008. Application-level isolation and recovery with solitude. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*. Eurosys '08. ACM, New York, NY, USA, 95–107.

JIM, T., SWAMY, N., AND HICKS, M. 2007. Defeating script injection attacks with browser-enforced embedded policies. In *Proceedings of the 16th international conference on World Wide Web*. WWW '07. ACM, New York, NY, USA, 601–610.

JOSHI, A., KING, S. T., DUNLAP, G. W., AND CHEN, P. M. 2005. Detecting past and present intrusions through vulnerability-specific predicates. In *Proceedings of the twentieth ACM symposium on Operating systems principles*. SOSP '05. ACM, New York, NY, USA, 91–104.

JOVANOVIC, N., KRUEGEL, C., AND KIRDA, E. 2006a. Pixy: A static analysis tool for detecting web application vulnerabilities (short paper). In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*. IEEE Computer Society, Washington, DC, USA, 258–263.

JOVANOVIC, N., KRUEGEL, C., AND KIRDA, E. 2006b. Precise alias analysis for static detection of web application vulnerabilities. In *Proceedings of the 2006 workshop on Programming languages and analysis for security*. PLAS '06. ACM, New York, NY, USA, 27–36.

KANICH, C., KREIBICH, C., LEVCHENKO, K., ENRIGHT, B., VOELKER, G. M., PAXSON, V., AND SAVAGE, S. 2008. Spamalytics: an empirical analysis of spam marketing conversion. In *Proceedings of the 15th ACM conference on Computer and communications security*. CCS '08. 3–14.

KIRDA, E., KRUEGEL, C., VIGNA, G., AND JOVANOVIC, N. 2006. Noxes: a client-side solution for mitigating cross-site scripting attacks. In *Proceedings of the 2006 ACM symposium on Applied computing*. SAC '06. ACM, New York, NY, USA, 330–337.

KRUEGEL, C. AND VIGNA, G. 2003. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*. CCS '03. ACM, New York, NY, USA, 251–261.

LAM, M. S., MARTIN, M., LIVSHITS, B., AND WHALEY, J. 2008. Securing web applications with static and dynamic information flow tracking. In *Proceedings of the 2008 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*. PEPM '08. ACM, New York, NY, USA, 3–12.

LAWTON, G. 2007. Web 2.0 creates security challenges. *Computer 40,* 10, 13 –16.

LI, C., JIANG, W., AND ZOU, X. 2009. Botnet: Survey and case study. In *Innovative Computing, Information and Control (ICICIC), 2009 Fourth International Conference on*. 1184 –1187.

LI, P., SALOUR, M., AND SU, X. 2008. A survey of internet worm detection and containment. *Communications Surveys Tutorials, IEEE 10,* 1, 20 –35.

LI, Z., TANG, Y., CAO, Y., RASTOGI, V., CHEN, Y., AND LIU, B. 2011. Webshield: Enabling various web defense techniques without client side modifications. In *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS)*.

LIN, K.-J. 2007. Building web 2.0. *Computer 40,* 5, 101 –102.

LU, L., YEGNESWARAN, V., PORRAS, P., AND LEE, W. 2010. Blade: an attack-agnostic approach for preventing drive-by malware infections. In *Proceedings of the 17th ACM conference on Computer and communications security*. CCS '10. ACM, New York, NY, USA, 440–450.

LUNT, T. F. 1993. A survey of intrusion detection techniques. *Computers & Security 12,* 4, 405 – 418.

MA, J., SAUL, L. K., SAVAGE, S., AND VOELKER, G. M. 2009. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '09. ACM, New York, NY, USA, 1245–1254.

MAGAZINIUS, J., ASKAROV, A., AND SABELFELD, A. 2010. A lattice-based approach to mashup security. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. ASIACCS '10. ACM, New York, NY, USA, 15–23.

MARHUSIN, M., CORNFORTH, D., AND LARKIN, H. 2008. An overview of recent advances in intrusion detection. In *Computer and Information Technology, 2008. CIT 2008. 8th IEEE International Conference on*. 432 –437.

MOSER, A., KRUEGEL, C., AND KIRDA, E. 2007. Exploring multiple execution paths for malware analysis. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*. SP '07. IEEE Computer Society, Washington, DC, USA, 231–245.

MOSHCHUK, A., BRAGIN, T., DEVILLE, D., GRIBBLE, S. D., AND LEVY, H. M. 2007. Spyproxy: execution-based detection of malicious web content. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*. USENIX Association, Berkeley, CA, USA, 3:1–3:16.

MOSHCHUK, E., BRAGIN, T., GRIBBLE, S. D., AND LEVY, H. M. 2006. A crawler-based study of spyware on the web. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*.

MOTOYAMA, M., LEVCHENKO, K., KANICH, C., MCCOY, D., VOELKER, G. M., AND SAVAGE, S. 2010. Re: Captchas: understanding captcha-solving services in an economic context. In *Proceedings of the 19th USENIX conference on Security*. USENIX Security'10. USENIX Association, Berkeley, CA, USA, 28–28.

MUKHERJEE, B., HEBERLEIN, L., AND LEVITT, K. 1994. Network intrusion detection. *Network, IEEE 8,* 3, 26 –41.

MURALI, A. AND RAO, M. 2005. A survey on intrusion detection approaches. In *Information and Communication Technologies, 2005. ICICT 2005. First International Conference on*. 233 – 240.

NAZARIO, J. 2009. Phoneyc: a virtual client honeypot. In *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*. LEET'09. USENIX Association, Berkeley, CA, USA, 6–6.

ORMEROD, T., WANG, L., DEBBABI, M., YOUSSEF, A., BINSALLEEH, H., BOUKHTOUTA, A., AND SINHA, P. 2010. Defaming botnet toolkits: A bottom-up approach to mitigating the threat. In *Emerging Security Information Systems and Technologies (SECURWARE), 2010 Fourth International Conference on*. 195 –200.

PENG, T., LECKIE, C., AND RAMAMOHANARAO, K. 2007. Survey of network-based defense mechanisms countering the dos and ddos problems. *ACM Comput. Surv. 39*.

POLYCHRONAKIS, M., ANAGNOSTAKIS, K. G., AND MARKATOS, E. P. 2007. Emulation-based detection of non-self-contained polymorphic shellcode. In *Proceedings of the 10th international conference on Recent advances in intrusion detection*. RAID'07. Springer-Verlag, Berlin, Heidelberg, 87–106.

POLYCHRONAKIS, M., MAVROMMATIS, P., AND PROVOS, N. 2008. Ghost turns zombie: exploring the life cycle of web-based malware. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*. USENIX Association, Berkeley, CA, USA, 11:1–11:8.

PROVOS, N., MAVROMMATIS, P., RAJAB, M. A., AND MONROSE, F. 2008. All your iframes point to us. In *Proceedings of USENIX Security Symposium*. 1–16.

PROVOS, N., MCNAMEE, D., MAVROMMATIS, P., WANG, K., AND MODADUGU, N. 2007. The ghost in the browser analysis of web-based malware. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. USENIX Association, Berkeley, CA, USA, 4–4.

QING, S. AND WEN, W. 2005. A survey and trends on internet worms. *Computers & Security 24,* 4, 334 – 346.

RATANAWORABHAN, P., LIVSHITS, B., AND ZORN, B. 2009. Nozzle: a defense against heap-spraying code injection attacks. In *Proceedings of the 18th conference on USENIX security symposium*. SSYM'09. USENIX Association, Berkeley, CA, USA, 169–186.

RBN Study 2007. Russian Business Network Study. http://www.bizeul.org/files/RBN_study.pdf.

REIS, C., DUNAGAN, J., WANG, H. J., DUBROVSKY, O., AND ESMEIR, S. 2006. Browsershield: vulnerability-driven filtering of dynamic html. In *Proceedings of the 7th symposium on Operating systems design and implementation*. OSDI '06. USENIX Association, Berkeley, CA, USA, 61–74.

REIS, C. AND GRIBBLE, S. D. 2009. Isolating web programs in modern browser architectures. In *Proceedings of the 4th ACM European conference on Computer systems*. EuroSys '09. ACM, New York, NY, USA, 219–232.

RFC-2828 2000. IETF RFC 2828. http://tools.ietf.org/html/rfc2828/.

RIECK, K., KRUEGER, T., AND DEWALD, A. 2010. Cujo: Efficient detection and prevention of drive-by-download attacks. In *Proceedings of Annual Computer Security Applications Conference 2010*. ACSAC '2010.

ROESCH, M. 1999. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX conference on System administration*. LISA '99. USENIX Association, Berkeley, CA, USA, 229–238.

RUBIN, A. AND GEER, D.E., J. 1998. A survey of web security. *Computer 31,* 9, 34 –41.

SABAHI, F. AND MOVAGHAR, A. 2008. Intrusion detection: A survey. In *Systems and Networks Communications, 2008. ICSNC '08. 3rd International Conference on*. 23 –26.

SABBOUH, M., HIGGINSON, J., SEMY, S., AND GAGNE, D. 2007. Web mashup scripting language. In *Proceedings of the 16th international conference on World Wide Web*. WWW '07. ACM, New York, NY, USA, 1305–1306.

SABELFELD, A. AND MYERS, A. C. 2003. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications 21*, 2003.

SADODDIN, R. AND GHORBANI, A. 2006. Alert correlation survey: framework and techniques. In *Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services*. PST '06. ACM, New York, NY, USA, 37:1–37:10.

SAXENA, P., AKHAWE, D., HANNA, S., MAO, F., MCCAMANT, S., AND SONG, D. 2010. A symbolic execution framework for javascript. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*. SP '10. IEEE Computer Society, Washington, DC, USA, 513–528.

SAXENA, P., HANNA, S., POOSANKAM, P., AND SONG, D. 2010. Flax: Systematic discovery of client-side validation vulnerabilities in rich web applications. In *In 17th Annual Network & Distributed System Security Symposium, (NDSS)*.

SCHMIDT, A.-D., SCHMIDT, H.-G., BATYUK, L., CLAUSEN, J., CAMTEPE, S., ALBAYRAK, S., AND YILDIZLI, C. 2009. Smartphone malware evolution revisited: Android next target? In *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*. 1–7.

SEIFERT, C., WELCH, I., AND KOMISARCZUK, P. 2009. Identification of malicious web pages through analysis of underlying dns and web server relationships.

SHABTAI, A., MOSKOVITCH, R., ELOVICI, Y., AND GLEZER, C. 2009. Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. *Inf. Secur. Tech. Rep. 14*, 16–29.

SIDDIQUI, M., WANG, M. C., AND LEE, J. 2008. A survey of data mining techniques for malware detection using file features. In *Proceedings of the 46th Annual Southeast Regional Conference on XX*. ACM-SE 46. ACM, New York, NY, USA, 509–510.

SIDIROGLOU, S., IOANNIDIS, J., KEROMYTIS, A., AND STOLFO, S. 2005. An email worm vaccine architecture. In *Information Security Practice and Experience*, R. Deng, F. Bao, H. Pang, and J. Zhou, Eds. Lecture Notes in Computer Science Series, vol. 3439. Springer Berlin / Heidelberg, 97–108.

SONG, C., ZHUGE, J., HAN, X., AND YE, Z. 2010. Preventing drive-by download via inter-module communication monitoring. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. ASIACCS '10. ACM, New York, NY, USA, 124–134.

SOTIROV, A. 2007. Heap feng shui in javascript. In *Proceedings of BlackHat Europe*.

SOTIROV, A. AND DOWD, M. 2008. Bypassing browser memory protections. In *Proceedings of BlackHat*.

SU, Z. AND WASSERMANN, G. 2006. The essence of command injection attacks in web applications. In *Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. POPL '06. ACM, New York, NY, USA, 372–382.

TIPTON, H. 2009. *Information Security Management Handbook, Volume 3* 6 Ed. CRC Press, Inc., Boca Raton, FL, USA.

Top Ten Project 2011. OWASP Top Ten Project. http://www.owasp.org/.

TOTH, T. AND KRUEGEL, C. 2002. Accurate buffer overflow detection via abstract payload execution. In *Proceedings of the 5th international conference on Recent advances in intrusion detection*. RAID'02. Springer-Verlag, Berlin, Heidelberg, 274–291.

VINOD, P., LAXMI, V., AND GAUR, M. 2009. Survey on malware detection methods. In *Proceedings of The Third Annual IIT Kanpur Hacker's Workshop 2009*.

WANG, H. J., GUO, C., SIMON, D. R., AND ZUGENMAIER, A. 2004. Shield: vulnerability-driven network filters for preventing known vulnerability exploits. *SIGCOMM Comput. Commun. Rev. 34*, 193–204.

WANG, Y., BECK, D., JIANG, X., AND ROUSSEV, R. 2006. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*.

WANG, Y., VERBOWSKI, C., DUNAGAN, J., CHEN, Y., WANG, H. J., AND YUAN, C. 2003. Strider: A black-box, state-based approach to change and configuration management and support. In *In Usenix LISA*. 159–172.

WASSERMANN, G. AND SU, Z. 2007. Sound and precise analysis of web applications for injection vulnerabilities. In *Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*. PLDI '07. ACM, New York, NY, USA, 32–41.

WASSERMANN, G. AND SU, Z. 2008. Static detection of cross-site scripting vulnerabilities. In *Proceedings of the 30th international conference on Software engineering*. ICSE '08. ACM, New York, NY, USA, 171–180.

WILHELM, J. AND CHIUEH, T.-C. 2007. A forced sampled execution approach to kernel rootkit identification. In *Proceedings of the 10th international conference on Recent advances in intrusion detection*. RAID'07. Springer-Verlag, Berlin, Heidelberg, 219–235.

XIE, Y. AND AIKEN, A. 2006. Static detection of security vulnerabilities in scripting languages. In *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15*. USENIX Association, Berkeley, CA, USA.

YOU, I. AND YIM, K. 2010. Malware obfuscation techniques: A brief survey. In *Proceedings of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications*. BWCCA '10. IEEE Computer Society, Washington, DC, USA, 297–300.

YUE, C. AND WANG, H. 2009. Characterizing insecure javascript practices on the web. In *Proceedings of the 18th international conference on World wide web*. WWW '09. ACM, New York, NY, USA, 961–970.

ZEIDANLOO, H., SHOOSHTARI, M., AMOLI, P., SAFARI, M., AND ZAMANI, M. 2010. A taxonomy of botnet detection techniques. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*. Vol. 2. 158 –162.

ZHU, Z., LU, G., CHEN, Y., FU, Z., ROBERTS, P., AND HAN, K. 2008. Botnet research survey. In *Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International*. 967 –972.