State of the Art: Automated Black-Box Web Application Vulnerability Testing

Jason Bau, Elie Bursztein, Divij Gupta, John Mitchell
Stanford University
Stanford, CA
{*jbau, divijg*}*@stanford.edu,* {*elie, mitchell*}*@cs.stanford.edu*

***Abstract*—Black-box web application vulnerability scanners are automated tools that probe web applications for security vulnerabilities. In order to assess the current state of the art, we obtained access to eight leading tools and carried out a study of: (i) the class of vulnerabilities tested by these scanners, (ii) their effectiveness against target vulnerabilities, and (iii) the relevance of the target vulnerabilities to vulnerabilities found in the wild. To conduct our study we used a custom web application vulnerable to known and projected vulnerabilities, and previous versions of widely used web applications containing known vulnerabilities. Our results show the promise and effectiveness of automated tools, as a group, and also some limitations. In particular, "stored" forms of Cross Site Scripting (XSS) and SQL Injection (SQLI) vulnerabilities are not currently found by many tools. Because our goal is to assess the potential of future research, not to evaluate specific vendors, we do not report comparative data or make any recommendations about purchase of specific tools.**

***Keywords*-Web Application Security; Black Box Testing; Vulnerability Detection; Security Standards Compliance;**

I. Introduction

Black-box web application vulnerability scanners are automated tools that probe web applications for security vulnerabilities, without access to source code used to build the applications. While there are intrinsic limitations of black-box tools, in comparison with code walkthrough, automated source code analysis tools, and procedures carried out by red teams, automated black-box tools also have advantages. Black-box scanners mimic external attacks from hackers, provide cost-effective methods for detecting a range of important vulnerabilities, and may configure and test defenses such as web application firewalls. Since the usefulness of black-box web scanners is directly related to their ability to detect vulnerabilities of interest to web developers, we undertook a study to determine the effectiveness of leading tools. Our goal in this paper is to report test results and identify the strengths of current tools, their limitations, and strategic directions for future research on web application scanning methods. Because this is an anonymized conference submission, we note that the authors of this study are university researchers.

Web application security vulnerabilities such as cross-site scripting, SQL injection, and cross-site request forgeries are acknowledged problems with thousands of vulnerabilities reported each year. These vulnerabilities allow attackers to perform malevolent actions that range from gaining unauthorized account access [1] to obtaining sensitive data such as credit card numbers [2]. In the extreme case, these vulnerabilities may reveal the identities of intelligence personnel [3]. Because of these risks, web application vulnerability remediation has been integrated into the compliance process of major commercial and governmental standards, e.g. the Payment Card Industry Data Security Standard (PCI DSS), Health Insurance Portability and Accountability Act (HIPAA), and the Sarbanes-Oxley Act. To meet these mandates, web application scanners that detect vulnerabilities, offer remediation advice, and generate compliance reports. Over the last few years, the web vulnerability scanner market as become a very active commercial space, with, for example, more than 50 products approved for PCI compliance [4].

This paper reports a study of current automated black-box web application vulnerability scanners, with the aim of providing the background needed to evaluate and identify the potential value of future research in this area. To the best of our knowledge this paper is the most comprehensive research on any group of web scanners to date. Because we were unable to find competitive open-source tools in this area (see Section VII), we contacted the vendors of eight well-known commercial vulnerabilities scanners and tested their scanners against a common set of sample applications. The eight scanners are listed in Table I. Our study aims to answer these three questions:

1) What vulnerabilities are tested by the scanners?
2) How representative are the scanner tests of vulnerability populations in the wild?
3) How effective are the scanners?

Because our goal is to assess the potential impact of future research, we report aggregate data about all scanners, and some data indicating the performance of the best-performing scanner on each of several measures. Because this is not a commercial study or comparative evaluation of individual scanners, we do not report comparative detection data or provide recommendations of specific tools. No single scanner is consistently top-ranked across all vulnerability categories.

We now outline our study methodology and summarize our most significant findings. We began by evaluating the

set of vulnerabilities tested by the scanners. Since most of the scanners provide visibility into the way that target vulnerability categories are scanned, including details of the distribution of their test vector sets by vulnerability classification, we use this and other measures to compare the scanner target vulnerability distribution with the distribution of in-the-wild web application vulnerabilities. We mine the latter from incidence rate data as recorded by VUPEN security [5], an aggregator and validator of vulnerabilities reported by various databases such as the National Vulnerability Database (NVD) provided by NIST [6]. Using database results, we also compare the incidence rates of web application vulnerability as a group against incidence rates for system vulnerabilities (e.g. buffer overflows) as group.

In the first phase of our experiments, we evaluate scanner detection performance on established web applications, using previous versions of Drupal, phpBB, and Wordpress, released around January 2006, all of which include well-known vulnerabilities. In the second phase of our experiments, we construct a custom testbed application containing an extensive set of contemporary vulnerabilities in proportion with the vulnerability population in the wild. Our testbed checks all of the vulnerabilities in the NIST Web Application Scanner Functional Specification [7] and tests 37 of the 41 scanner vulnerability detection capabililities in the Web Application Security Consortium [8] evaluation guide for web application scanners. (See Section VII). Our testbed application also measures scanner ability to understand and crawl links written in various encodings and content technologies.

We use our custom application to measure elapsed scanning time and scanner-generated network traffic, and most importantly, we tested the scanners for vulnerability detection and false positive performance.

Our most significant findings include:

1) The vulnerabilities for which the scanners test most extensively are, in order, Information Disclosure, Cross Site Scripting (XSS), SQL Injection, and other forms of Cross Channel Scripting (XCS). This testing distribution is roughly consistent with the vulnerability population in the wild.
2) Although many scanners are effective at following links whose targets are textually present in served pages, most are not effective at following links through active content technologies such as Java applets, SilverLight, and Flash.
3) The scanners as a group are effective at detecting well-known vulnerabilities. They performed capably at detecting vulnerabilities already reported to VuPen from historical application versions. Also, the scanners detected basic "reflected" cross-site scripting well, with an average detection rate of over 60%.
4) The scanner performed particularly poorly at detecting "stored" vulnerabilities. For example, no scanner detected any of our constructed second-order SQLI vulnerabilities, and the stored XSS detection rate was only 15%. Other limitations are discussed further in this paper.

Table I
STUDIED VULNERABILITY SCANNERS

Company	Product	Version	Scanning Profiles Used
Acunetix	WVS	6.5	Default and Stored XSS
Cenzic	HailStorm Pro	6.0	Best Practices, PCI Infrastructure, and Session
HP	WebInspect	8.0	All Checks
IBM	Rational AppScan	7.9	Complete
McAfee	McAfee SECURE	Web	Hack Simulation and DoS
N-Stalker	QA Edition	7.0.0	Everything
Qualys	QualysGuard PCI	Web	N/A
Rapid7	NeXpose	4.8.0	PCI

Our analysis suggests room for improvement in detecting vulnerabilities inserted in our testbed, and we propose potential areas of research in Section VIII. However, we have made no attempt to measure the financial value of these tools to potential users. Scanners performing as shown may have significant value to customers, when used systematically as part of an overall security program. In addition, we did not quantify the relative importance of detecting specific vulnerabilities. In principle, a scanner with a lower detection rate may be more useful if the smaller number of vulnerabilities it detects are individually more important to customers.

Section II of this paper discusses the black box scanners and their vulnerability test vectors. Section III establishes the population of reported web vulnerabilities. Section IV presents scanner results on Wordress, phpBB, and Drupal versions released around January 2006. Section V discusses testbed results by vulnerability category for the aggregated scanner set and also false positives. Section VI contains some remarks by scanner, on individual scanner performance as well as user experience. Section VII discusses related work and section VIII concludes by highlighting research opportunities resultant from this work.

II. BLACK BOX SCANNERS

We begin by describing the general usage scenario and software architecture of the black-box web vulnerability scanners. We then discuss the vulnerability categories which they aim to detect, including test vector statistics where available. Table I lists the eight scanners incorporated in our study, which include products from several of the most-established security companies in the industry. All the scanners in the study are approved for PCI Compliance testing [4]. The prices of the scanners in our study range from hundreds to tens-of-thousands of dollars. Given such a wide price range and also variations in usability, potential customers of the scanners would likely not make a purchase decision on detection performance alone.

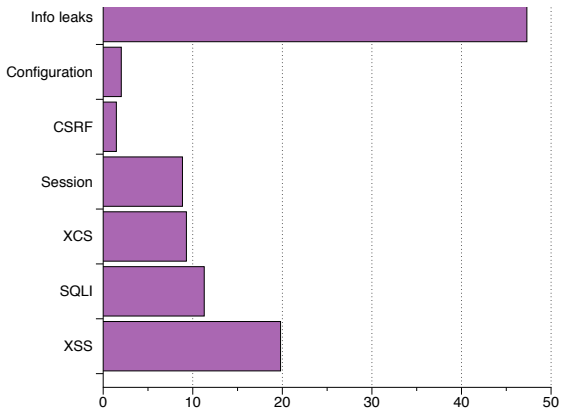


Figure 1. Scanner Test Vector Percentage Distribution

A. Usage Scenario

To begin a scanning session using a typical scanner, the user must enter the entry URL of the web application as well as provide a single set of user login credentials for this application. The user then must specify options for the scanner's page crawler, in order to maximize page scanning coverage. Most scanners tested allow a "crawl-only" mode, so that the user can verify that the provided login and the crawler options are working as expected. After setting the crawler, the user then specifies the the scanning profile, or test vector set, to be used in the vulnerability detection run, before launching the scan. All scanners can proceed automatically with the scan after profile selection, and most include interactive modes where the user may direct the scanner to scan each page. In our testbed experiments, we always set the scanner to run, in automated mode, the most comprehensive set of tests available, to maximize vulnerability detection capability.

B. Software Architecture Descriptions

We ran two of the tested scanners, McAfee and Qualys, as remote services whereby the user configures the scanner via a web-interface before launching the scan from a vendor-run server farm. The other six scanners were tested as software packages running on a local computer, although the NeXpose scanner runs as a network service accessed by browser via an IP port (thus naturally supporting multiple scanner instances run by one interface). All scanners, as would be expected of black box web-application testers, generate http requests as test vectors and analyze the http response sent by the web server for vulnerabilities. All local scanner engines seem to run in a single process, except for the Cenzic scanner, which runs a separate browser process that appears to actually render the http response in order to find potential vulnerabilities therein.

Table II
CONSENSUS VULNERABILITY CLASSIFICATION ACROSS SCANNERS

Classification	Example Vulnerability
Cross-Site Scripting (XSS)	Cross-Site Scripting
SQL Injection (SQLI)	SQL Injection
Cross Channel Scripting	Arbitrary File Upload Remote File Inclusion OS Command Injection Code Injection
Session Management	Session Fixation Session Prediction Authentication Bypass
Cross-Site Request Forgery	Cross Site Request Forgery
SSL/Server Configuration	SSL Misconfiguration Insecure HTTP Methods
Information Leakage	Insecure Temp File Path Traversal Source Code Disclosure Error Message Disclosure

C. Vulnerability Categories Targeted by Scanners

As each scanner in our study is qualified for PCI compliance, they are mandated to test for each of the Open Web Application Security Project (OWASP) Top Ten 2007 [9] vulnerability categories. We also examine the scanning profile customization features of each scanner for further insight into their target vulnerability categories. All scanners except Rapid7 and Qualys allow views of the scanning profile by target vulnerability category, which are often direct from the OWASP Top Ten 2007 and 2010rc1, Web Application Security Consortium (WASC) Threat Classification version 1 [10], or the Common Weakness Enumeration (CWE) top 25 [11]. In fact, each of the six allow very fine-grained test customization, resulting in a set of over 100 different targeted vulnerability categories, too numerous to list here. However, when related vulnerability categories were combined into more general classifications, we were able to find a set of consensus classifications for which all tools test. Table II presents this list of consensus classifications, along with some example vulnerabilities from each classification. We have kept Cross-Site Scripting and SQL Injection as their own vulnerability classifications due to their preponderant rate of occurrence (supported by "in the wild" data in the next section) and their targeting by all scanners. The Cross Channel Scripting classification [12] includes all vulnerabilities, including those listed in the table, allowing the user to inject code "across a channel" onto the web server that executes on the server or a client browser, aside from XSS and SQLI.

D. Test Vector Statistics

We were able to obtain detailed enough test profile information for four scanners (McAfee, IBM, HP, and Acunetix) to evaluate how many test vectors target each vulnerabilities classification, a rough measure of how much "attention"

scanner vendors devote to each classification. Figure 1 plots the percentage of vectors targeting each classification aggregated over the four scanners. The results show that scanners devote most testing to information leakage vulnerabilities, followed by XSS and SQLI vulnerabilities.

III. Vulnerability Population from VuPen-Verified NVD

In order to evaluate how well the vulnerability categories tested by the scanners represent the web application vulnerability population "in the wild", we took all of the web vulnerability categories forming the consensus classifications from Table II and performed queries against the VUPEN Security Vulnerability Notification Service database for the years 2005 through 2009. We chose this particular database as our reference as it aggregates vulnerabilities, *verifies* them through the generation of successful attack vectors, and reports them to sources such as the Common Vulnerabilities and Exposures (CVE) [13] feed of the National Vulnerability Database.

We collected from the VUPEN database the relative incidence rate trends of the web application vulnerability classes, which are plotted in Figure 2. Figure 3 plots incidences of web application vulnerabilities against incidences of system vulnerabilities, e.g. Buffer Overflow, Integer Overflow, Format String, Memory Corruption, and Race Conditions, again collected by us using data from VUPEN.

Figure 2 demonstrates that Cross-Site Scripting, SQL Injection, and other forms of Cross-Channel Scripting have consistently counted as three of the top four reported web application vulnerability classes, with Information Leak being the other top vulnerability. These are also the top four vulnerability classes by scanner test vector count. Within these four, scanner test vectors for Information Leak amount to twice that of any other vulnerability class, but the Information Leak incidence rates in the wild are generally lower than that of XSS, SQLI, and XCS. We speculate that perhaps test vectors for detecting information leakage, which may be as simple as checking for accessible common default pathnames, are easier to create than other test types. Overall, however, it does appear that the testing emphasis for black-box scanners as a group is reasonably proportional to the verified vulnerability population in the wild.

We believe that the increase in SSL vulnerabilities shown in figure 2 does not indicate a need for increased black-box scanning. A large number of SSL vulnerabilities were reported in 2009, causing the upward trend in SSL incidences. However, these are actually certificate spoofing vulnerabilities that allow a certificate issued for one domain name, usually containing a null-character, to become valid for another domain name [14], [15]. As this vulnerability is caused by mistakes made by the certificate authority and the client application (usually browser), it cannot be prevented by the website operator and thus cannot be detected by web application scanning. In effect, the number of SSL/Server configuration vulnerabilities that web application scanners may reasonably aim to detect does not appear to increase with the increased SSL vulnerability incidence rate.

Finally, Figures 2 and 3 suggest that 2006 was a particularly high-incident year for web application vulnerabilities, with incidents actually decreasing in subsequent years. (This trend is also confirmed by searches in the CVE database.) While it is impossible to be certain, evidence gathered during the course of this study, including the effectiveness of the scanners at detecting basic XSS and SQLI vulnerabilities, suggests that the decrease may possibly be attributable to headway made by the security community against these vulnerabilities. Improved security, however, has been answered in turn by efforts to uncover more novel forms of the vulnerabilities.

Table III
Previously-Reported vs Scanner-Found Vulnerabilities For Drupal, phpBB2, and Wordpress

Category	Drupal 4.7.0		phpBB2 2.0.19		Wordpress 1.5strayhorn	
	Known	Found	Known	Found	Known	Found
XSS	6	2	5	2	13	7
SQLI	2	1	1	1	8	4
XCS	4	0	1	0	8	3
Session	5	4	4	4	6	5
CSRF	2	0	1	0	1	1
Info Leak	4	3	1	1	6	4

IV. Scanner Results on Common Web Applications

Having confirmed that the testing vector distribution of black-box web vulnerability scanners as a group roughly correlates with the vulnerability population trends in the wild, we now examine whether the scanners are actually successful at finding existent vulnerabilities. We ran all scanners on three popular web applications, Drupal, phpBB2, and Wordpress, all with known vulnerabilities. We chose to scan application versions released around January 2006, as this was prior to the peak in vulnerability reports in 2006. While these are field applications with some inherent uncertainty as to their exact vulnerability content, the early release dates mean these application versions are the most field-tested, with most vulnerabilities likely to have been recorded by VUPEN via the NVD.

Table III lists the specific application versions tested as well as the number of known vulnerabilities, including those reported by the VUPEN database for each of these versions. For all applications, we installed only the default modules and included no add-ons.

Table III also shows the number of vulnerabilities found by *any* scanners in the group, out of the set of known

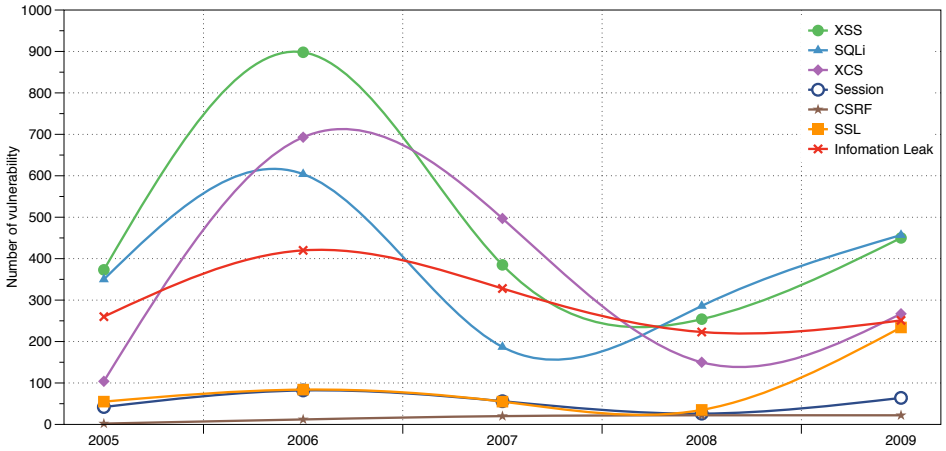


Figure 2. Comparison of Web Application Vulnerability Classes in VUPEN Database

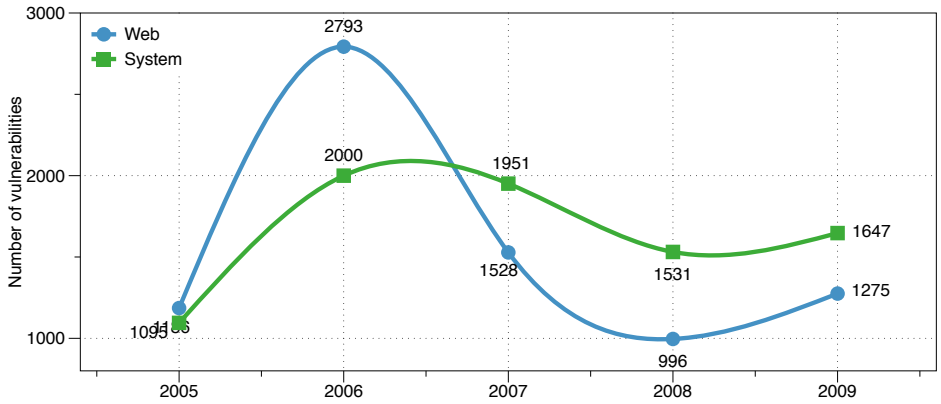


Figure 3. Web Application Vulnerabilities versus System Vulnerabilities in VUPEN Database

vulnerabilities. As the table shows, the scanner in total did a generally good job of detecting these previously known vulnerabilities. They did particularly well in the Information Disclosure and Session Management classifications, leading to the hypothesis that effective test vectors are easier to add for these categories than others. The scanners also did a reasonable job of detecting XSS and SQLI vulnerabilities, with about 50% detection rate for both. The low detection rate in the CSRF classification may possibly be explained by the small number of CSRF test vectors. Anecdotally, one scanner vendor confirmed that they do not report CSRF vulnerabilities due to the difficulty of determining which forms in the application require protection from CSRF.

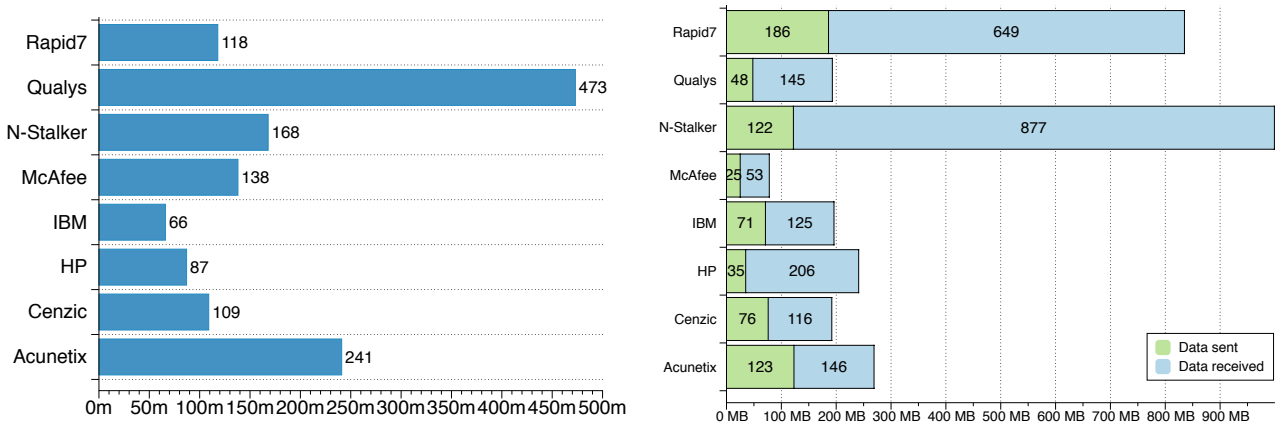
V. Scanner Results on Custom Testbed

In addition to testing scanner detection performance on established web applications, we also evaluated the scanners in a controlled environment. We developed our own custom testbed application containing hand-inserted vulnerabilities, each of which have a proven attack pattern. We verified each of the vulnerabilities present in this environment, allowing us significantly smaller uncertainty in vulnerability content than in the case of field-deployed applications. (The scanners as a group did not uncover any unintended vulnerabilities in our web application.) We plan to release this testbed publically.

For each vulnerability classification, we incorporated both "textbook" instances and also forward-looking instances, such as XSS with non-standard tags, for each vulnerability classification. However, we kept the vulnerability content of our testbed fairly proportional with the vulnerability population in the wild.

Our testbed has around 50 unique URLs and around 3000 lines of code, installed on a Linux 2.6.18-128.1.6.el5 server running Apache 2.2.3, MySQL 5.0.45, and PHP 5.1.6. PhpMyAdmin was also running on our server alongside the testbed application, solely for administrative purposes; we thus ignored any scanner results having to do with phpMyAdmin.

The remainder of this section is devoted to scanner testbed data. We begin by presenting the performance footprint of each scanner on our testbed. Following this, we report page coverage results, designed to test scanner understanding of



(a) Scanner Execution Time in Minutes

(b) Scanner Bytes Sent and Received

Figure 4. Scanner Footprint

various content technologies. We then present vulnerability detection results, first an overview and subsequently by vulnerability classification, giving a brief overview of our testbed design for each classification. Finally, we discuss false positives, including experimentally designed "traps" for false positives as well as scanner results.

A. Scanner Time and Network Footprint

Figures 4a and 4b respectively plot the time required to scan the testbed application and the number of network bytes sent/received by each scanner, as measured on the web server by tcpdump. Scanning time ranged from 66 to 473 minutes, while network traffic ranged from 80 MB to nearly 1 GB.

Perhaps surprisingly, the scanning time and network traffic statistics seem to be relatively independent of each other, as exemplified by the Rapid7, Qualys, N-Stalker, and McAfee results. It is interesting that the two remote services, Qualys and McAfee, generated comparatively low amounts of network traffic. Finally, we wish to note that the footprint statistics are not indicative of vulnerability detection performance.

B. Coverage Results

To experimentally evaluate site coverage, we wrote hyperlinks using the technology in each category shown in figure 5 and embedded each landing page with tracker code that measured whether the link was followed. For Java, SilverLight, and Flash, the linked applet or movie is a simple, bare shell containing only the hyperlink. We then link to the technology page containing the link from the application home page, which is written in regular php.

The link encoding category encompasses links written in hexadecimal, decimal, octal, and html encodings, with the landing page file named in regular ASCII. The "POST link" test involves a link that only shows up when certain selections are made on a POST form. The other technologies are self explanatory. Figure 5 shows the experimental results, where the measure is percentage of successful links crawled over total existent links by technology category.

Figure 5 shows that the scanners as a group have fairly low comprehension of active technologies such as Java applets, SilverLight, and, surprisingly given its widespread use, Flash. We speculate that some scanners only perform textual analysis of http responses in order to collect URLs, thus allowing them to perform decently on script-based links, which are represented in text, but not allowing them to follow links embedded in compiled objects such as Java applets and Flash movies. This would also explain the better coverage of SilverLight over Flash and Java, as SilverLight is delivered in a text-based markup language. We also see that the scanners could improve their understanding of various link encodings.

C. Vulnerability Detection Results

1) Overall Results: Figure 6 presents by vulnerability classification the vulnerability detection rate averaged over all scanners. The detection rate is simply calculated as the number of vulnerabilities found over the (known) number of total vulnerabilities. Results for each vulnerability classifications, including an added malware detection classification, are explained in detail in individual sub-sections to follow. Each vulnerability classification sub-section describes the testbed for the category, plots the *average* detection rate over all scanners, and also plots anonymous individual scanner results for the category *sorted from best- to worst-performing for that category*.

The results show that the scanners as a group are fairly effective at detecting basic "reflected" cross-site scripting (XSS type 1), with a detection rate of over 60%. Also, although not shown, basic forms of first-order SQL Injection were detected by a majority of scanners. Unfortunately, the overall results for the first-order SQL vulnerability

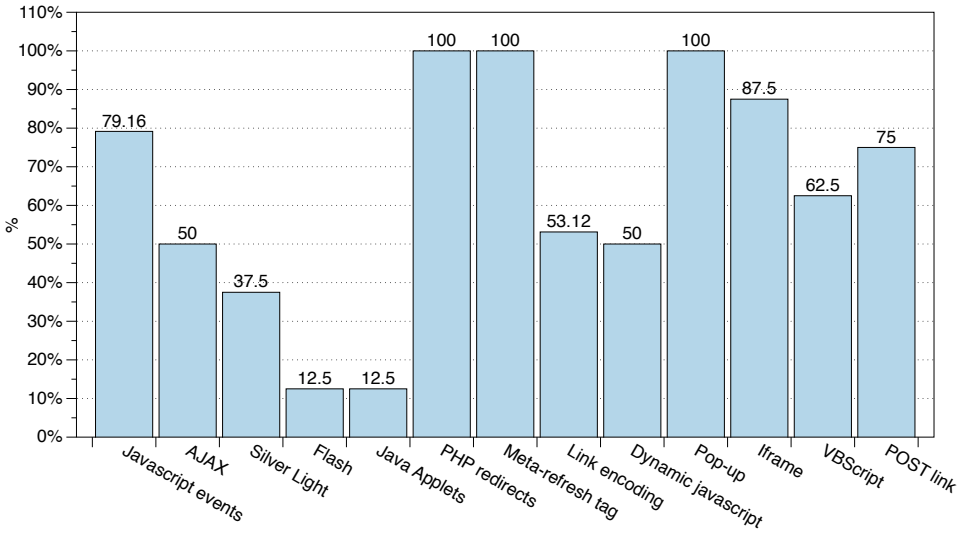


Figure 5. Successful Link Traversals over Total Links by Technology Category, Averaged Over All Scanners.

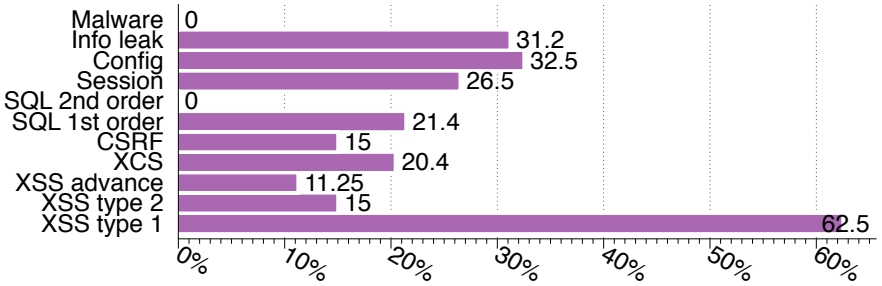


Figure 6. Average Scanner Vulnerability Detection Rate By Category

classification were dragged down by poor scanner detection of more complex forms of first-order SQL injection that use different keywords. Aside from the XSS type 1 classification, there were no other vulnerability classifications where the scanners as a group detected more than 32.5% of the vulnerabilities. In some cases, scanners were unable to detect testbed vulnerabilities which were an exact match for a category listed in the scanning profile. We also note how poorly the scanners performed at detecting "stored" vulnerabilities, i.e. XSS type 2 and second-order SQL injection, and how no scanner was able to detect the presence of malware. We will discuss our thoughts on how to improve detection of these under-performing categories in Section VIII.

2) Cross-Site Scripting: Due to the preponderance of Cross-Site Scripting vulnerabilities in the wild, we divided Cross-Site Scripting into three sub-classes: XSS type 1, XSS type 2, and XSS advanced. XSS type 1 consists of textbook examples of reflected XSS, performed via the `<script>` tag. XSS type 2 consists of stored XSS vulnerabilities, where un-sanitized user input is written to the database and later performs scripting when read from the database. XSS advanced encompasses novel forms of reflected and stored XSS, using non-standard tags and keywords, such `<style>` and `prompt()` [16], or using alternative scripting technologies such as Flash. For XSS advanced tests using novel keywords, we filtered out any user inputs that did not contain the appropriate keywords.

As previously mentioned, Figure 7 shows that the scanners performed decently well on XSS type 1, with all scanners detecting at least 50% of the vulnerabilities. For the other categories, however, the scanners performed poorly as a group, with only the leading performer detecting more than 20% of vulnerabilities, and numerous scanners failing to detect any vulnerabilities.

3) SQL Injection: We also divided SQL Injection vulnerabilities into two sub-classes, first-order and second-order, for the same reason as dividing the XSS classification. First-order SQL Injection vulnerabilities results in immediate SQL command execution upon user input submission, while second-order SQL Injection requires unsanitized user input to be loaded from the database. The first-order SQL vulnerability classification also includes both textbook vulnerabilities as well as vulnerabilities dependent on non-standard keywords such as `LIKE` and `UNION`, where user inputs not

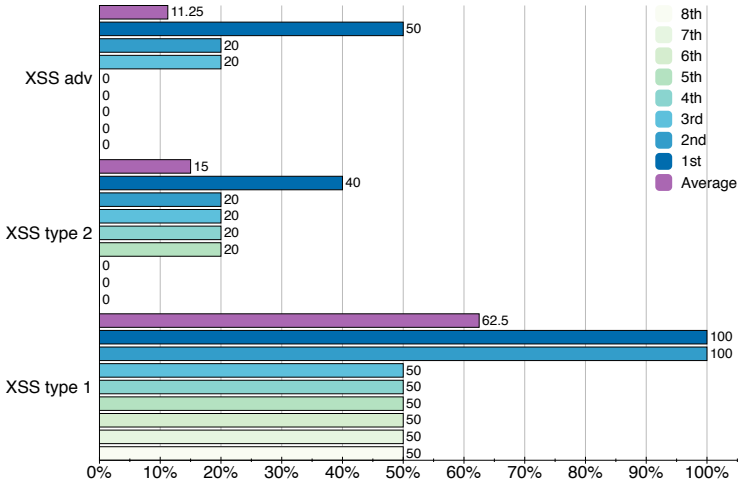


Figure 7. XSS Detection Results Sorted By Scanner Rank in Category

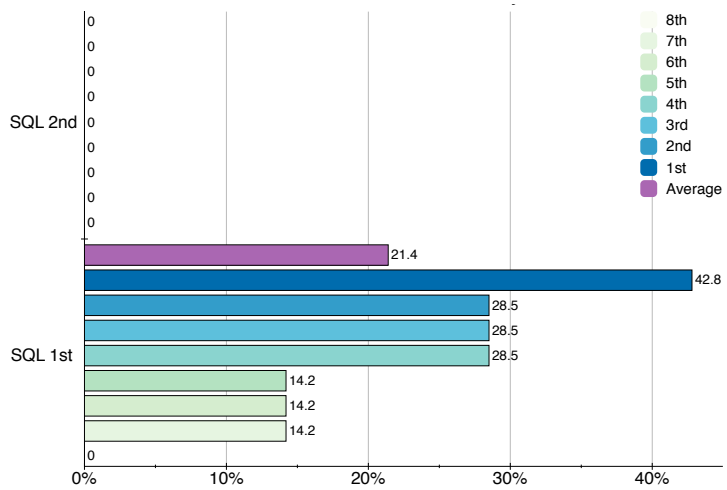


Figure 8. SQL Injection Detection Results Sorted by Scanner Rank in Category

containing the appropriate keyword are again filtered out. Also, in all cases, malformed SQL queries in our testbed result in a displayed SQL error message, so scanners do not have to rely on blind SQL Injection detection.

Figure 8 shows that 7 of 8 scanners were able to detect the basic first-order SQL injection vulnerability (accounting for the 14.2%), but only one scanner was able to exceed 40% rate of detection for all first-order SQL injection vulnerabilities. Similar to XSS results, second-order SQLI vulnerability detection is significantly worse than first-order, with no scanner able to detect even one such vulnerability.

4) Cross-Channel Scripting: As described in a previous section, the Cross Channel Scripting classification includes all vulnerabilities allowing the attacker to inject code onto the web server that manipulates the server or a client browser. In our testbed, this classification included vulnerabilities in XPath injection, Malicious File Upload, Open Redirects, Cross-Frame Scripting, Server Side Includes, Path Traversal, Header Injection (HTTP Response Splitting), Flash Parameter Injection, and SMTP Injection.

As Figure 9 demonstrates, the scanners as a group performed fairly poorly on this class of vulnerabilities. Only Server-Side Includes and Path Traversal were detected by a majority of scanners.

5) Session Management: The Session vulnerability classification include session management flaws as well as authentication and cookie flaws. The testbed authentication vulnerabilities include credentials being sent over unencrypted HTTP, auto-complete enabled in the password field, submitting sensitive information over GET requests, weak password and password recovery questions, and weak registration CAPTCHAs. The session management and cookie vulnerabilities include insecure session cookies, non-HttpOnly cookies, too broad cookie path restrictions, pre-

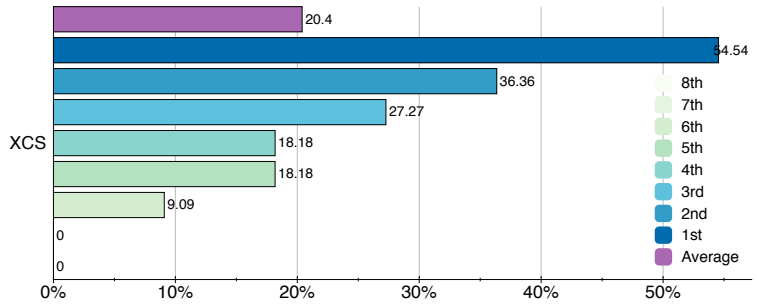


Figure 9. Cross-Channel Scripting Detection Results Sorted by Scanner Rank in Category

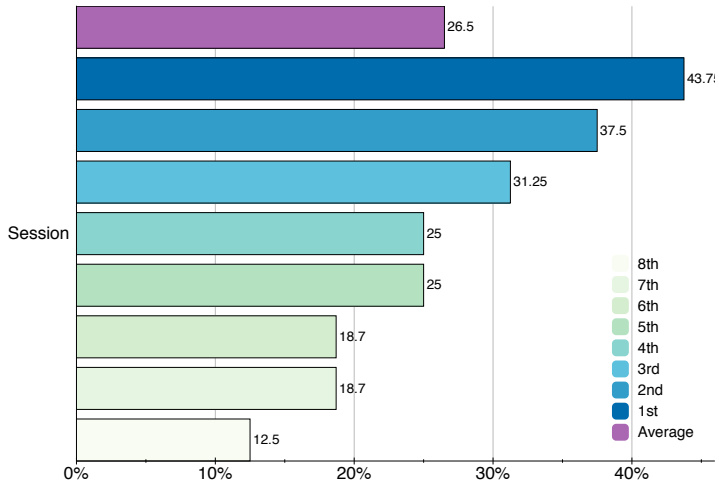


Figure 10. Session Management Vulnerability Detection Results Sorted by Scanner Rank in Category

dictable session and authentication id values, session fixation, ineffective logout, mixed content pages, and caching of sensitive content.

As a group, the scanners performed better at detecting this vulnerability class than the SQLI, XSS, and XCS classes. Figure 10 shows that scanner performance is fairly evenly distributed between the best performer at 43.7% detection and the worst at 12.5% detection.

6) Cross-Site Request Forgery: Nearly all forms on our testbed application do not use any sort of randomized authentication token, making them vulnerable to Cross-Site Request Forgery. However, we only considered as requiring CSRF protection the forms which are only available after login. Our testbed contains post-login forms without any authorization token and also post-login form which utilize tokens with very few bits of entropy. In addition to the CSRF vulnerabilities just mentioned, our testbed also included session tokens that do not reset after form submission, GET-method forms vulnerable to CSRF, and CSRF-like JSON hijacking vulnerabilities.

Results in Figure 11 show that two scanners fared relatively well at CSRF detection, each achieving 40% detection rates. On the other extreme, four scanners detected none of the vulnerabilities in this classification, with one vendor confirming that they did not report CSRF vulnerabilities at the time of testing.

7) Information Disclosure: Our testbed application leaks sensitive information regarding SQL database names via the `die()` function and existent user names via AJAX requests. Backup source code files are also left accessible, and path disclosure vulnerabilities are also present.

Figure 12 shows that this was one of two vulnerability categories where the scanners as a group performed the best. A majority of scanners detected all of the backup file disclosures, as well as the path disclosure vulnerabilities.

8) Server and Cryptographic Configuration: While server and cryptography vulnerabilities do not technically occur at the application layer, they affect web application security all the same and should be detected by the vulnerability scanners. Our server configuration contained improper PHP setting in the 'open_basedir' and 'allow_url_fopen' variables and allowed the HTTP TRACE request. The SSL of the server was also mis-configured, with a self-signed SSL certificate and weak cipher strength.

Figure 13 shows that this was the other of the two vulnerability categories where the scanners as a group

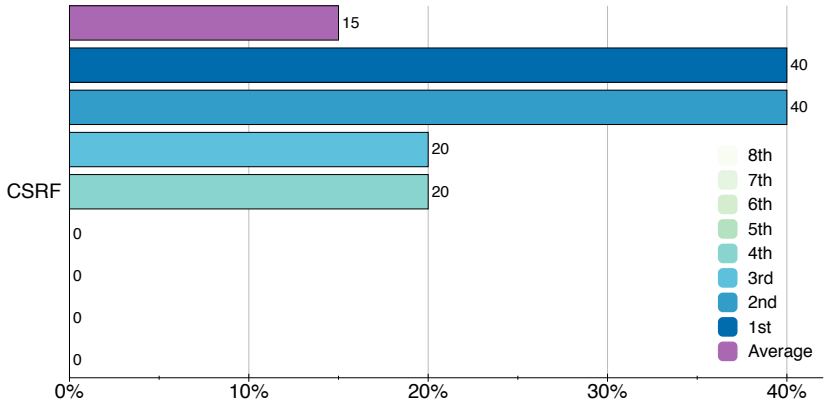


Figure 11. Cross-Site Request Forgery Detection Results Sorted by Scanner Rank in Category

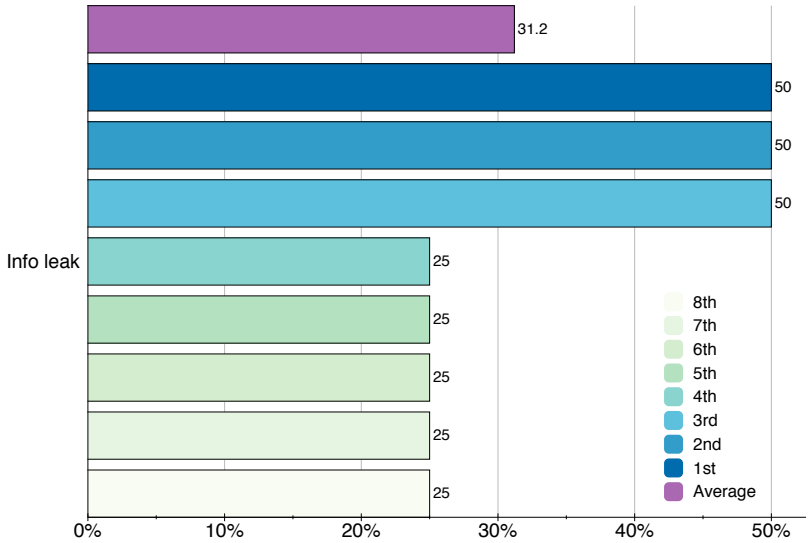


Figure 12. Information Disclosure Detection Results Sorted by Scanner Rank in Category

performed the best. Seven of eight scanners detected the TRACE request vulnerability. On the other hand, we were surprised that less than half of the scanners detected the self-signed SSL certificate, as this seems to be a simple check to prevent significant impact on the user-experience of the site, especially with newer browsers.

9) Detection of Malware: Finally, we decided to add the detection of malware as a category in the scanner testbed. With the proliferation of open-source code, it is uncertain whether website operators are familiar with the entirety of the codebase which operates their site. Malware detection serves as a check that websites are not unwitting partners aiding malicious third-parties in exploiting their users. This feature is also useful as a defense-in-depth measure in the case that attackers have succeeded in injecting malicious code onto the site. Thus, we inserted simple pieces of malware on our testbed site, a javascript keystroke logger at the login page and a malicious file in the user-content section. However, as Figure 6 demonstrates, no scanner reported the presence of any malware on our testbed site.

D. False Positive Results

We designed two potential "traps" for false positives in our testbed. The first involves using javascript `alert()` as intended site behavior, to see if any scanner would wrongly classify this as a sign of a Cross-Site Scripting vulnerability. The second involves inserting user input into the right-hand side of a Javascript string variable assignment within a `<script>` block but not doing anything else with that variable. Any quotation symbols in the user input that would create a genuine XSS vulnerability are html-encoded by the testbed, and all ';' characters are filtered. This in effect creates a benign region within a `<script>` block that reflects user input, which can trap scanners simply searching for user-manipulable text within script blocks. The "`alert()`" trap did not cause any false positives, but

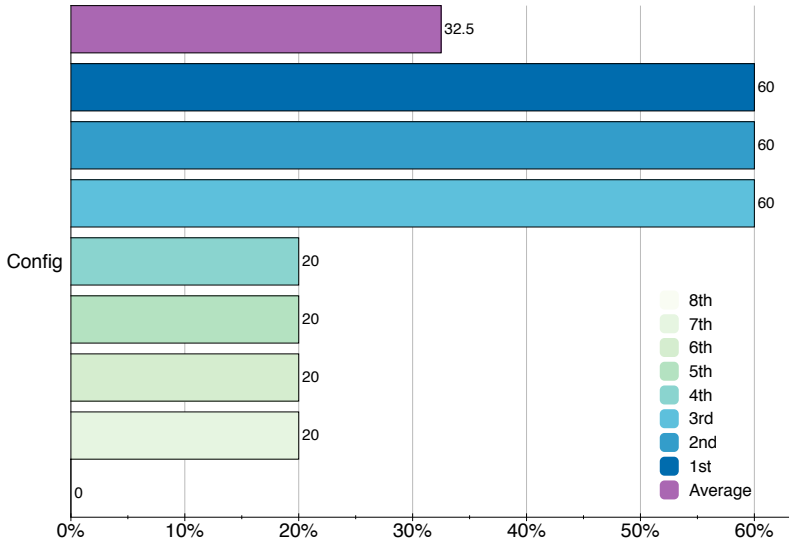


Figure 13. Server and Cryptographic Configuration Vulnerability Detection Results Sorted by Scanner Rank in Category

the benign region within a `<script>` block caused false positives in two scanners–one reporting the false positive in a single URL and the other in 13 different URLs.

Figure 14 plots the number of false positives reported by each scanner in sorted order for this category. For reference, there are around 90 total confirmed vulnerabilities in our testbed. It is noteworthy that several scanners reported no false positives, and that some of the scanners with low false-positives also had among the highest vulnerability detection rates. The two scanners with the highest number of false positive, both with vulnerability detection rates among the lowest, reported numerous accessible code backup files where none existed. The worst performing scanner for false positives also reported false file inclusion, SQL Injection, IP disclosure, path disclosure, and forms accepting POST parameters form GET requests. This scanner also classifies hidden form values as vulnerabilities, contradicting established practices for CSRF prevention using hidden form authentication tokens. Among all other scanners, the only other false positives of note are a CSRF vulnerability reported despite the presence of an authentication token, and auto-complete being reported for a password field where it was actually turned-off.

Finally, some scanners emit general warnings when they detect a potential vulnerability, such as a GET form method or a form without hidden authentication fields, without actually pinpointing the URL of the offending forms. We counted these as detections in our data-collection methodology, but, given the general nature of these warnings, could have just as easily listed them as false positives.

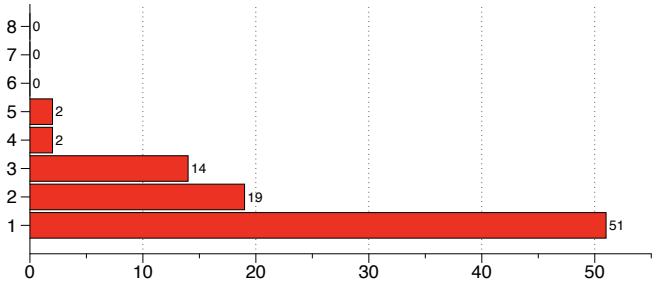


Figure 14. False Positive Count in Sorted Order By Scanner

VI. Experimental and Scanner Usage Observations

We have thus far focused primarily on the detection performance of the scanners as a group of different vulnerability classifications. In this section, we will remark on some by-scanner characteristics, without making overall comparative rankings of one product versus another.

We observed that no individual scanner was a top-performer in every vulnerability classification. Often, scanners with a leading detection rate in one vulnerability category lagged in other categories. For example, the leading scanner in both the XSS and SQL Injection categories was among the bottom three in detecting Session Management vulnerabilities, while the leader for Session Vulnerabilities lagged in XSS and SQLI. This leads us to believe that scanner vendors may benefit as a community from a cross-vendor-pollination of ideas.

Reiterating briefly from the false positive results, we did find that scanners with high detection rates were able to effectively control false positives, and that scanners with low detection rates could produce many false positives.

Additionally, there were reticent scanners that reported few total vulnerabilities, making both detection rate and false positive count low.

The remote versus local distinction in scanner architecture makes for interesting choices for the scanner customer. In our experimental experience, remote scanners were convenient as they experienced no install and system compatibility issues, offered portable user-interface and report-storage, ran with complete automation after the initial test configuration, and did not consume internal network and system resources. On the other hand, commercial enterprises may be concerned with privacy and information disclosure issues resulting from conducting scans over a public network. In response to customer preferences, some vendors, such as Cenzic and Rapid7, offer scanners both as a remote service and a local software package. In our experiments, we observed that the architecture (local versus remote) of the scanners did not appear to be a factor in overall detection results.

We also wish to remark on the user experience of the scanners, specifically regarding test automation. Most tools offer a choice between interactive and automated scanning modes, and we anticipate that, given the expected run-time of the scanners, many users will select the automated mode. On a particular tool, however, even the automated mode requires user-interaction to dismiss javascript `alert()` boxes, ironically inserted by the tool's XSS test vectors. This caused workflow disruption in our laboratory environment, so we expect that it would carry over when scanning larger, deployed applications.

Finally, we wish to note that while the vulnerability detection rates reported in this paper are generally less than 50%, this fact by itself should not be considered an indictment against the usefulness of automated black-box scanners. Black-box scanners may in fact prove to be very useful components in security-auditing programs upon more detailed consideration of factors such as cost and time saved from manual review.

VII. Related Work

Much regulatory and industry effort has been devoted to vulnerability categorization. The Common Vulnerabilities and Exposures database [13] (CVE) feed of the NVD, sponsored by the US Dept. of Homeland Security, associates each vulnerability in its database with a Common Weakness Enumeration (CWE) category [11], which include but are not limited to web application categories. Industry web application security special interest groups OWASP [17] and WASC [18] have published web vulnerability-specific classifications in their Top Ten [9] and [10] projects respectively. WASC has also published a report on web-vulnerability statistics [19], with vulnerability and detection rate data sourced from automated black-box scanner, manual black-box penetration testing, and white-box security auditing vendors. The vulnerability statistics they report are supportive of our results, but their self-reported detection rates are in general higher than our rates, since manual white-box security audits, which have higher reported detection rates, are also included in the study.

In addition, NIST [7] and WASC [8] have published evaluation criteria for web application scanners. We consulted these public categorizations and scanner evaluation guides to ensure the comprehensiveness of our testbed. Our testbed checks all of the recommendations in the NIST guide and 37 of the 41 first-and-second-level testing capability listed by WASC.

Almost all academic research on tools for web application security has been source code analysis, with a focus on detecting XSS and SQLI via information flow, modeling checking analysis, or a combination thereof. Work by Wassermann [20], Lam [21], Kieżun [22], Jovanovic [23], and Huang [24] all fall into this category.

Kals et. al. [25] and McAllister et. al. [26] implemented automated black box web vulnerability scanners, with the former targeting SQLI and XSS vulnerabilities and the latter utilizing user interactions to generate more effective test cases targeting reflected and stored XSS. Maggi et. al. [27] discuss techniques to reduce false positive counts in automated intrusion detection, which is applicable to black-box scanning. Interesting open-source scanner projects include W3AF [28] and Powerfuzzer [29], which we evaluated but did not include in the study due to their lack of testing for authentication and server vulnerabilities, and Nikto [30], which in counterpoint to W3AF and PowerFuzzer focuses on server vulnerabilities instead of user-input validation.

In terms of testbeds for black-box web application vulnerability scanners, there are a number of "vulnerability demonstration sites", such as WebGoat by OWASP [31], Hacme Bank [32], and AltoroMutual [33], that offer vulnerability education for website developers as well as sales-demonstration for scanner product capabilities. Due to their well-known status and/or intended purpose, we did not evaluate any of the scanners on these sites as we did not view the site as independent testbeds. However, Suto [34] has produced an interesting comparison of seven black-box scanners by running the products against several of these demonstration sites. Finally, Fonseca et. al. [35] evaluated the XSS and SQLI detection performance of three anonymous commerical application via automated software fault-injection methods.

VIII. Conclusion

We studied the vulnerabilities that current black-box scanners aim to detect and their effectiveness in detecting these vulnerabilities. Our survey of web-application vulnerabilities in the wild shows that Cross-Site Scripting, SQL Injection, other forms of Cross-Channel Scripting, and Information Disclosure are the most prevalent classes of vulnerabilities.

Further, we found that black-box web application vulnerability scanners do, in general, expend testing effort in rough proportion to the vulnerability population in the wild. As shown by our experimental results on previous versions of popular applications and textbook cases of Cross-Site Scripting and SQL Injection, black-box scanners are adept at detecting straightforward historical vulnerabilities.

On the other hand, black-box scanner detection rates show room for improvement in other classes of vulnerabilities, such as advanced and second-order forms of XSS and SQLI, other forms of Cross-Channel Scripting, Cross-Site Request Forgery, and Malware Presence. Deficiencies in the CSRF and Malware classifications, and possibly in XCS, may simply be attributable to lack-of-emphasis in vendor test suites. Low detection rates in advanced and second-order XSS and SQLI may indicate more systematic flaws, such as insufficient storage modeling in XSS and SQLI detection. Indeed, multiple vendors confirmed their difficulty in designing tests which detect second-order vulnerabilities. Although our data suggests room for improvement in detecting vulnerabilities, the scanners we tested may have significant value to customers, when used systematically as part of an overall security program.

The strongest research opportunities lie in detecting advanced and second-order forms of XSS and SQLI, because these forms of vulnerabilities are prevalent (and will continue to be) and tools do not currently perform well, despite significant effort. There are several ways that scanner performance might be improved in the "advanced" vulnerability categories, which consist of attacks using novel and non-standard keywords. A reactive approach is to develop more nimble processes of converting newly discovered vulnerabilities into appropriate test vectors. A more foundational approach could involve modeling application semantics in a meaningful way.

For 2nd order XSS and SQLI vulnerabilities, one natural research problem is to increase observability. The scanners have difficulty confirming that a script or code injection into storage was successful and also may have trouble linking a later observation with the earlier injection event. We can anecdotally confirm the latter statement, as one of the tools succeeded in injecting a stored Javascript `alert()` but later failed to identify this as a stored XSS. Thus, we believe that detection rates may be improved by better tool understanding of the application database model. More basic scanner modifications such as adding a second user login for observing cross-account stored vulnerabilities and better management of observational passes after the initial injection pass should also improve detection results in these categories.

As far as site coverage, the low coverage results for SilverLight, Flash and Java Applets and the false positives triggered by the "benign" Javascript trap lead us to suggest another area of improvement for black-box scanners: better understanding of active content and scripting languages.

Acknowledgment

The authors would like to thank Acunetix, Cenzic, IBM, McAfee, Qualys, and Rapid7 for their participation in this study.

References

[1] StrongWebmail CEO's mail account hacked via XSS. ZDNet. [Online]. Available: http://blogs.zdnet.com/security/?p=3514

[2] D. Litchfield. SQL Injection and Data Security Breaches. [Online]. Available: http://www.davidlitchfield.com/blog/archives/00000001.htm

[3] Websites of WHO and MI5 Hacked Using XSS Attacks. Spamfigher.com. [Online]. Available: http://tinyurl.com/yfqauzo

[4] Approved Scanning Vendors. Payment Card Industry Security Standards Council. [Online]. Available: https://www.pcisecuritystandards.org/pdfs/asv_report.html

[5] VUPEN Security. [Online]. Available: http://www.vupen.com

[6] National Vulnerability Database. Dept. of Homeland Security National Cyber Security Division. [Online]. Available: http://web.nvd.nist.gov

[7] *Software Assurance Tools: Web Application Security Scanner Functional Specification*, National Institute of Standards and Technology Std., Rev. 1.0.

[8] Web Application Security Scanner Evaluation Criteria. Web Application Security Consortium. [Online]. Available: http://projects.webappsec.org/Web-Application-Security-Scanner-Evaluation-Criteria

[9] OWASP Top Ten Project. Open Web Application Security Project. [Online]. Available: http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[10] Web Security Threat Classification. Web Application Security Consortium. [Online]. Available: http://www.webappsec.org/projects/threat/

[11] Common Weakness Enumeration. [Online]. Available: http://cwe.mitre.org

[12] H. Bojinov, E. Bursztein, and D. Boneh, "Xcs: cross channel scripting and its impact on web applications," in *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2009, pp. 420–431.

[13] Common Vulnerabilities and Exposures. [Online]. Available: http://cve.mitre.org

[14] D. Kaminsky, "Black Ops of PKI," *BlackHat USA*, August 2009.

[15] M. Marlinspike, "More Tricks For Defeating SSL," *BlackHat USA*, August 2009.

[16] E. V. Nava and D. Lindsay, "Our Favorite XSS Filters and How to Attack Them," *BlackHat USA*, August 2009.

[17] Open Web Application Security Project. [Online]. Available: http://www.owasp.org

[18] Web Application Security Consortium. [Online]. Available: http://www.wasc.org

[19] Web Application Security Statistics. Web Application Security Consortium. [Online]. Available: http://projects.webappsec.org/Web-Application-Security-Statistics

[20] G. Wassermann and Z. Su, "Sound and precise analysis of web applications for injection vulnerabilities," *SIGPLAN Not.*, vol. 42, no. 6, pp. 32–41, 2007.

[21] M. S. Lam, M. Martin, B. Livshits, and J. Whaley, "Securing web applications with static and dynamic information flow tracking," in *PEPM '08: Proceedings of the 2008 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*. New York, NY, USA: ACM, 2008, pp. 3–12.

[22] A. Kieżun, P. J. Guo, K. Jayaraman, and M. D. Ernst, "Automatic creation of SQL injection and cross-site scripting attacks," in *ICSE'09, Proceedings of the 30th International Conference on Software Engineering*, Vancouver, BC, Canada, May 20–22, 2009.

[23] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: A static analysis tool for detecting web application vulnerabilities (short paper)," in *2006 IEEE Symposium on Security and Privacy*, 2006, pp. 258–263. [Online]. Available: http://www.iseclab.org/papers/pixy.pdf

[24] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, and S.-Y. Kuo, "Securing web application code by static analysis and runtime protection," in *WWW '04: Proceedings of the 13th international conference on World Wide Web*. New York, NY, USA: ACM, 2004, pp. 40–52.

[25] S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic, "Secubat: a web vulnerability scanner," in *WWW '06: Proc. 15th Int'l Conf. World Wide Web*, 2006, pp. 247–256.

[26] S. Mcallister, E. Kirda, and C. Kruegel, "Leveraging user interactions for in-depth testing of web applications," in *RAID '08: Proc. 11th Int'l Symp. Recent Advances in Intrusion Detection*, 2008, pp. 191–210.

[27] F. Maggi, W. K. Robertson, C. Krügel, and G. Vigna, "Protecting a moving target: Addressing web application concept drift," in *RAID*, 2009, pp. 21–40.

[28] Web Application Attack and Audit Framework. [Online]. Available: http://w3af.sourceforge.net/

[29] Powerfuzzer. [Online]. Available: http://www.powerfuzzer.com/

[30] CIRT.net Nikto Scanner. [Online]. Available: http://cirt.net/nikto2

[31] WebGoat Project. OWASP. [Online]. Available: http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project

[32] HacmeBank. McAfee Corp. [Online]. Available: http://www.foundstone.com/us/resources/proddesc/hacmebank.htm

[33] AltoroMutual Bank. Watchfire Corp. [Online]. Available: http://demo.testfire.net/

[34] Larry Suto. Analyzing the Accuracy and Time Costs of Web Application Security Scanners. [Online]. Available: http://ha.ckers.org/files/Accuracy_and_Time_Costs_of_Web_App_Scanners.pdf

[35] J. Fonseca, M. Vieira, and H. Madeira, "Testing and comparing web vulnerability scanning tools for sql injection and xss attacks," *Pacific Rim Int'l Symp. Dependable Computing, IEEE*, vol. 0, pp. 365–372, 2007.