Open Source Web Vulnerability Scanners: The Cost Effective Choice?

Kinnaird McQuade
kinnaird_mcquade@marymount.edu
Information Technology Department
Marymount University
Arlington, VA 22207

Abstract

A plethora of tools are available to software testers so that software vulnerabilities can be mitigated before product deployment. However, some of these tools are less effective than others. In particular, open source dynamic web vulnerability scanners raise concerns including (1) total attack and input vector support, (2) scan coverage of different application protocols, and (3) rate of required manual detection versus automated detection. Additionally, what is often most attractive about proprietary scanners is vendor support and frequent software maintenance bundled with a paid licensing agreement. Indeed, the need for software support will ensure the longevity of proprietary dynamic web vulnerability scanners on the market. However, a low-cost alternative is available and recommended for web developers involved in agile development at small to medium sized development firms; it is the finding of this research that when a combination of certain open source tools are used in conjunction with a specific scanning strategy, there is a greater vulnerability detection accuracy than solely using a single proprietary scanner.

Keywords: web vulnerability scanners, application security, dynamic tools, software development lifecycle, quality assurance, open source

1. INTRODUCTION

Serious security vulnerabilities that an attacker can exploit to take control over a website, compromise user accounts, or access sensitive data occur across every major industry in the United States. During 2013, 96% of all web applications tested by a major application security firm had at least one serious security vulnerability, and the medium number of vulnerabilities per website was 14 (Cenzic, p. 3, 2014). Hackers are concentrating their efforts on web applications where users enter sensitive information such as forms, login pages, and shopping carts to gain access into servers to find personal and corporate information. Insecure web applications jeopardize corporate and government databases that are critical to the financial health and economic stability of the United States. The sooner a security vulnerability is discovered and corrected, the less likely that error will create more errors that may result in significantly more effort to correct.

Application security specialists use a combination of static, dynamic, and manual testing techniques to perform security assessments (OWASP, p.4, 2009). Reliance on one technique for security testing before the deployment phase in the software development lifecycle would be a potentially disastrous mistake. Therefore, application security firms usually implement a holistic assessment using these techniques as well as performing architecture risk analysis (Stevens, 2011). In order to understand the role of dynamic web vulnerability scanners, it is important to understand the difference between dynamic, static, and manual testing.

To properly explain the difference between dynamic and static testing: in the case example of a "Hello World" Java application, static tools would only analyze the Main.java class at the program compile time, whereas dynamic tools would not only scan the Main.java class, but would also scan the entirety of the accompanying Java Runtime Environment. Gartner explains that dynamic scanners analyze applications "in their running state during operation or testing phases," simulating prepared attacks and analyzing the response to determine the existence of vulnerabilities (MacDonald & Feiman, 2014).

Burp Suite Professional is an excellent case example of manual testing (Portswigger Web Security, 2014). The web browser forwards all incoming requests to Burp Proxy, which intercepts the HTTPS traffic passing in either direction and allows the user to analyze and alter the GET and POST requests to profile or tamper with the web application. Manual testing requires a notable amount of training and technical knowledge (Halfond, Choudhary, & Orso, pg. 18, 2011).

There are certainly limitations to the usefulness of dynamic web vulnerability scanners; they are not the cure-all solution for vulnerability detection. Cigital estimates that dynamic testing only uncovers up to 12% of all discovered software flaws, whereas Architecture Risk Analysis can uncover up to 60% of discoveries (Stevens, 2011). Amongst the assessment techniques for detecting web application vulnerabilities, dynamic scanning tools are much less effective when large-scale security testing occurs just prior to the deployment phase. However, it has an indispensable role in the security of the session, presentation, and application layers of web applications and should be used in conjunction with secure design principles, static testing, and manual code review. Brian W. Kernighan and Rob Pike explain in *The Practice of Programming (1999)*:

> *It's tedious and unreliable to do much testing by hand; proper testing involves lots of tests, lots of inputs, and lots of comparisons of outputs. Testing should therefore be done by programs, which don't get tired or careless.*

Mark G Graff and Kenneth R. van Wyk advise that performing static, dynamic, and manual testing at every stage of the development cycle is paramount to securing the session, presentation, and application layers of a web application (Graff & van Wyk, p. 157, 2003). The automation of these scanners increases the time efficiency – and consequently, the cost-effectiveness – of vulnerability detection – while offering the promise of allowing developers to focus on coding and design rather than having to become nearly full-time security testers. These scanners can be used in a local environment before deployment. They should be used in conjunction with static scanners, which is not included in the scope of this paper. With training and an understanding of the underlying behavior in Web Application Vulnerability Scanners, software developers in agile development groups can perform security testing between development cycles so that software errors are discovered earlier, without sacrificing detection accuracy.

Although the liberty of choice when selecting a web vulnerability scanner (or any software, for that matter) upon budget flexibility, the scanner's vulnerability detection features, accuracy, coverage, and stability should be considered just as strongly as the reality of financial pressures. Indeed, the persistence of severe software bugs can prove to be a much higher cost than commercial licenses – but making the best decision for an organization involves much more than just asking the question – "to spend or not to spend."

In any purchase, high cost can deceive the consumer into assuming that the product is of higher quality. There are high-quality commercial scanners on the market, but the fact is that there is no silver bullet solution in this area of Information Technology or any other. Black-box vulnerability scanners do not cover all features identified for comparison purposes by WAVSEP. Proprietary Web Vulnerability Scanners such as IBM Appscan and HP WebInspect, when used in conjunction, cover the most categories. However, they can also cost the user $20,000 per year and $10,000 a year per installation, respectively. To complicate the issue, their detection accuracy on vulnerable applications in the 6 main vulnerabilities evaluated by WAVSEP does not always have the highest detection accuracy.

After analyzing and confirming his findings, this author of this paper confirmed that a low-cost combination of vulnerability scanning tools can be used to support the same attack vectors with a detection accuracy that is greater than or equal to proprietary scanners. The finding of this paper is most highly recommended for agile developers wishing to test their modules for security vulnerabilities before entering another iteration cycle.

2. LITERATURE REVIEW

The existing research that will be discussed in this section examines the cost-effectiveness of different Web Application Vulnerability Scanners, the limitations of these dynamic scanners, and perspective regarding the commercial methods of providing product evaluation for potential customers.

Analyzing the Accuracy and Time Costs of Web Application Security Scanners

In Larry Suto's follow-up to his 2007 paper, he generalized the web application testing community into two groups. He explains that Group One uses scanners in a "point and shoot" manner, relies on the scanner's crawler and automation to exercise the site's functionality within minimal or no human guidance, while Group Two believes that scanners should be an adjunct to human testing and only used to grab the easy vulnerabilities, or "low hanging fruit" (Suto, p. 13, 2010).

Why Johnny Can't Pentest: An Analysis of Black-box Web Vulnerability Scanners

In their 2010 work, Doupe, Cova, and Vigna explain that web application scanners are essentially comprised of three main modules: a crawler, an attacker, and an analysis module (Doupe et. al, p. 3, 2010). During their research, they created the WackoPicko Web Site – a purposefully vulnerable picture sharing site – which is now distributed on OWASP's Broken Web Applications Virtual Machine. All of their Web Application Vulnerability Scanners were commercial tools, with the exception of Grendel-Scan, w3af, and Paros. Their research generated three strong conclusions:

(1) Support for well-known, pervasive technology (such as JavaScript, Flash, etc.) should be improved.
(2) More sophisticated algorithms are needed to perform "deep" crawling and track the state of the application under test.
(3) More research is warranted to automate the detection of application logic vulnerabilities.

Similarly to Suto's 2010 paper, they assert that vulnerability scanner technology is "far from being point-and-click tools to be used by anybody" and that "web application black-box security scanners require a sophisticated understanding of the application under test and of the limitations of the tool in order to be effective" (Doupe et. al., p. 20, 2010). Their final conclusion is that there is "no strong correlation between the cost of the scanner and functionality provided as some of the free or very cost-effective scanners performed as well as scanners that cost thousands of dollars" (Doupe et. al., p. 20, 2010)

The most interesting part about their well-earned conclusion is their perspective that an intimate understanding of the target application is mandatory in order to achieve useful results. It is important that developers are involved in the dynamic testing process for maximum effectiveness.

Avoiding the Test Site Fallacy

Veracode, a Massachusetts-based application security company known for their static code analysis engine, produced a white paper in 2012 that sought to explain why purchasers of proprietary web vulnerability scanners "should not gauge the abilities and effectiveness of a particular scanner by only looking at the results from scanning public test sites" (Dawson, 2012). In their evaluation, they addressed five sites for analysis, produced by a few prominent scanner vendors. The test sites for IBM Appscan, NTOSpider, HP WebInspect, and Acunetix were included in their evaluation. Veracode criticized these closed-source sites for possessing unrealistic or fake vulnerabilities, unrealistic form validation or checks, missing vulnerability coverage, and for hiding these issues from the user. From their own testing of the site, they provide convincing evidence of these shortcomings from each individual site.

Although Veracode's alarming attack on the vendors of proprietary scanners raises many questions about the true capabilities of these scanners, it does not lessen the industry-wide attraction to proprietary scanners. It does,

however, indicate the need for third-party evaluations providing comparative analysis on the scanners in question.

3. EXISTING EVALUATION METHODS

One of the most popular technologies regarding Web Application Vulnerabilities is the Broken Web Apps Project, produced by the OWASP Foundation and distributed as a Linux Virtual Machine. It is a collection of training applications, intentionally vulnerable applications, outdated versions of real vulnerable applications, applications for testing tools only (including WAVSEP), and vulnerability demonstration pages. The most well-known of these web apps is WebGoat, an application security training web app that offers over 30 lessons dealing with various vulnerabilities.

Gartner's Magic Quadrant for Application Security Testing offers a qualitative evaluation of the commercial vendors in the Web Application Security Market. Their evaluation criteria focuses on the support structure between the scanner's vendor and their customer, drawing their evidence from the collection of surveys from customers, responses from vendors, and hundreds of inquiries regarding the scanners throughout 2013-2014. Customer experience, marketing strategy, and completeness of company vision were among the comparison points. This report is especially helpful for companies who intend on purchasing a license and expect to use this software frequently.

The Web Application Vulnerability Scanner Evaluation Project (WAVSEP) is a yearly benchmark produced by Shay Chen, a widely respected application security researcher, which tests the vulnerability detection accuracy of 63 Black Box Web Application Vulnerability Scanners and discusses their capabilities. In addition to the benchmark, Shay Chen has also published a feature comparison between all the scanners, the numbers and types of Vulnerability Detection Features, and the detection accuracy of 6 software weakness types (5 of which are in the OWASP Top 10). One of the most helpful work products from WAVSEP for consumers is the visual, understandable comparison of all the scanners so that the consumer can use the data for their own analysis.

Web Application Security Consortium produced a set of evaluation criteria in 2008 (WASSEC) that grades web application scanners on their ability to effectively test web applications and identify vulnerabilities. It covers areas such as crawling, parsing, session handling, testing, and reporting. The goal, similar to Shay Chen's WAVSEP project, is a vendor-neutral document produced to:

(1) Provide scanner users with the tools they need for conducting a detailed evaluation and making an informed decision about which web application scanner(s) to choose;
(2) Provide scanner developers with a list of capabilities to compare their tools against to help them create a roadmap of future enhancements.

4. MAPPING THE SUBSTITUTIONS

The first requirement for determining the proper combination of scanners is that the maximum amount of input vectors and attack vectors should be covered so that most of the target application can be scanned. The recommended combination of open source substitutions has been mapped in Appendix 2.

The following tools are recommended as a result of this evaluation:

1. Burp Suite Professional - $300 per year; low-cost proprietary.
2. IRONWASP – Open Source
3. Zed Attack Proxy (ZAP) – Open Source
4. Arachni – Open Source
5. W3af – Open Source.

It should be noted that w3af was not included for evaluation in this study. Its detection accuracy in the WAVSEP evaluation was equal to or less than the detection accuracy of the highest-performing open source scanners in each vulnerability category. It is, however, a highly regarded web vulnerability scanner in the open source penetration testing community. Its exploitation features after vulnerability scanning are more extensible than some of the recommended tools in this study. However, this study focused on vulnerability detection, not exploitation. It was decided for this reason to not include w3af in the results of this paper.
IRONWASP is a powerful open source web vulnerability scanner that offers plugin compatibility with both Python and Ruby, making it an attractive open source scanner for

those wishing to make their own scanners or customize the tool with their own plugins (Kuppan, 2014). It is very easy to use and offers some interactive scan capabilities that are similar to Burp Suite.

Zed Attack Proxy (ZAP) is noticeably similar to Burp Suite Professional and doubles as a scanner with the help of the Plug-n-Hack Mozilla Firefox plugin (OWASP Zed Attack Proxy Project, 2014). The design is clean, the tool is lightweight, and it is strong in its detection accuracy. It is not as automated as the Arachni web user interface and does not have the cleanest reporting tool, but it is clearly a powerful open source scanner

Arachni is a very high-performing, easy-to-use, modular web vulnerability scanner written in Ruby (Laskos, 2014). It offers a web user interface that permits multiple users to manage scans and collaborate afterwards. It is very quick and its HTML report presentation is very clean. The web user interface is very customizable but unfortunately the interactive capabilities for manual testing are reserved for the command line version only. However, the available scan options are so customizable that it would be ideal for a software developer wishing for a "point and click" dynamic scanning experience.

There are multiple attack vectors not supported by the recommended combination of tools in this study. However, these attack vectors are either rarely implemented issues or are so vendor-specific that manual and static testing is the only reliable or realistic way to find a vulnerability. As indicated by the visually significant gap in input vector support (see Appendix 1), most proprietary scanners do not even support these areas.

The data below highlights his findings in six areas:

(1) Old, Backup, and Unreferenced Files (BACKUP)
(2) Path traversal/Local File Inclusion (LFI)
(3) Unvalidated Redirects (REDIRECT)
(4) Reflected Cross Site Scripting (RXSS)
(5) SQL Injection (SQLi)
(6) Remote File Inclusion (RFI)

These first three are the areas where open source stands out above the proprietary software in detection accuracy; in the last group of three, open source programs are equal in detection accuracy.

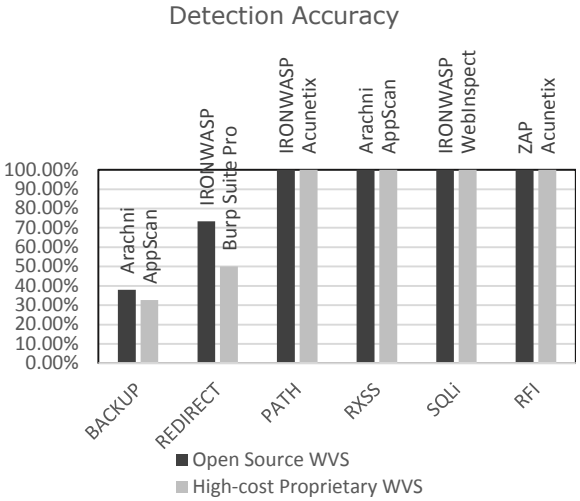

Figure 1. Results adapted from WAVSEP.

A visual representation of the vulnerability detection from the WAVSEP 2014 benchmark can be seen in Figure 1 above.

WAVSEP Detection Accuracy by tool

	Open Source or Low-cost scanner	High-cost Proprietary scanner
Remote File Inclusion	100% Arachni	100% IBM Appscan
SQL Detection	100% SQLMap, Burp Suite	100% Acunetix WebInspect
Reflected XSS	100% IRONWASP	100% Acunetix, AppScan, Netsparker
Path Traversal/LFI	100% Arachni	100% AppScan
Unvalidated Redirect	73.33% IRONWASP	50% WebInspect
Old, Backup, and Unreferenced Files	38.04% ZAP	32.61% Acunetix

Figure 2. Results adapted from WAVSEP.

The recommended open source combination performs well in each one of these vulnerability detection results. In the last four vulnerability groups, open source tools have the same detection accuracy as their proprietary counterparts – 100%. Surprisingly, open source

outperforms the highest performing proprietary scanners in the first two categories – Unreferenced Backup files and Unvalidated Redirects.

The detection accuracy for these vulnerabilities can be seen above in Figure 2. The detection accuracy is listed in the same cell as the scanner name, subdivided by a dashed line. The recommended open source tools are indicated by the grey column.

The WAVSEP v1.5 evaluation "emulates different **common** test case scenarios for generic technologies." Shay Chen explains his reasoning for the WAVSEP evaluation structure further in his 2014 evaluation:

> *A scanner that is not accurate enough will not be able to identify many exposures, and might classify non-vulnerable entry points as vulnerable. These tests aim to assess how good is each tool at detecting the vulnerabilities it claims to support, in a supported input vector, which is located in a known entry point, without any restrictions that can prevent the tool from operating properly (Chen, 2014).*

To put it simply – while each of these scanners can detect more flaws than the vulnerabilities that were evaluated, the test results speak to the overall accuracy of the tool itself when the tests are put on an equal playing field. To support this point, this paper includes an independent evaluation of the recommended combination of tools below using the Java Enterprise Edition 7 evaluation.

5. EXPERIMENTAL EVALUATION

This research used Duke's Forest, a Web Application provided by Oracle as part of their Java EE 7 tutorial, to evaluate the effectiveness of the recommended combination of free or low-cost web vulnerability scanners. Duke's Forest is an e-commerce application provided by Oracle as a case study for understanding the full capabilities of Java Enterprise Edition 7 (Jendrock, et. al, 2014). This final example in the Java EE 7 Tutorial acts as a store that provides gardening supplies to online shoppers. The complete web application includes a product catalog, customer self-registration, and a shopping cart; it also offers shipment and payment functionalities that can be managed through a separate administrative portal.

Design

Duke's Forest is a simple e-commerce application that interacts with the user through the Duke's Store interface. A non-administrative user of Duke's Store is able to browse the product catalog for foresting supplies, add items to their shopping cart, specify the shipment process, and manage their basic account information.

Authentication and Security

Duke's Forest uses HTTP Basic Authentication and JAAS (Java Authentication and Authorization Service) to authenticate the user. Security constraints are built in to differentiate between customers and administrators. Single Sign-On (SSO) is used to simplify the administrator's browsing experience in navigating between the Duke's Store and Duke's Shipment portals.

Browsing the product catalog

The foresting product categories include Plants, Food, Services, and Tools. Users can browse the product catalog, filter their searches according to product categories, and view product details.

Signing up as a new customer

While the product catalog can be browsed without user registration, Duke's Store requires user registration in order to add items to the shopping cart and make purchases. The password value must be at least 7 characters in length. There are already two user names included in the database upon website deployment – jack@example.com and robert@example.com – and they both have the password, 1234.

Shopping Cart and Checking Out

A registered user on Duke's Store can purchase foresting supplies. The purchase follows a process in which a shopping cart is filled with the foresting items to be purchased. After the foresting supplies are added to the cart, the sum price is calculated, and the order is placed. Orders exceeding $1,000 are not permitted because the Payment web service denies orders over that limit. Once the order is placed, the user must wait for the administrative user to approve the shipping process.

Viewing order status

After the checkout process is complete, a message appears to the user: "Your order is being processed. Check the Orders page to see the status of your order."

Administrative Interface
Duke's Forest allocates a portion of the website for administrators only. The Administrator page is used to perform back office operations, such as the creation, editing, updating, and deletion of products, categories, customers, and user groups.

Vulnerabilities
Cross-Site Scripting: The ImageServlet.java class in the /com/forest/web/util package is vulnerable to Cross-Site Scripting (OWASP, 2014c). Almost every other java class in the program utilizes the data validation method provided by Java Enterprise Edition, but this particular class simply does not include the mechanism. This could be due to an error by the programmer.

SQL Injection: the JSESSIONID cookie, which is displayed in cleartext in the URL, is vulnerable to SQL Injection attacks (OWASP, 2014a). By appending a basic blind SQL injection statement such as 1' OR '1'='1 as encoded URL into the cookie, different responses are returned, indicating that the input is being incorporated into the SQL query incorrectly. Upon further manual testing, this can be exploited to gain access to more database information.

Cleartext Credentials: The store page, login page, customer registration page, and index.html page contain a login form which is submitted over clear-text HTTP headers. Manipulating the GET and POST requests with an intercepting proxy such as Zed Attack Proxy or Burp Suite Proxy can disclose passwords to a hacker performing a man-in-the-middle attack such as ARP poisoning (OWASP, 2014b).

Insecure Cookies: Forms with sensitive content, like passwords, must be sent over HTTPS. The login page, strangely, has enabled the HTTP-only flag for the web cookie – exposing any legitimate user to credential theft, man-in-the-middle attacks, and cross-site request forgery (OWASP, 2014b).

Session token in URL: The session ID is transmitted in the URL when the user logs in to the website in certain cases. In the case of a man-in-the-middle attack, an attacker could use the session ID to create a fake browser cookie in order to gain administrator or user access to the system (OWASP, 2014b).

Password Auto-enabled: Disabling the autocomplete function in the login form will limit the user's browser from remembering user credentials that are entered into HTML forms. The stored credentials can be captured by an attacker who gains physical or remote access to the computer. If the administrator's computer were hacked, for example, the security of the company's website would be jeopardized (OWASP, 2014b).

Missing anti-Cross-Site Request Forgery token: There is no synchronized token mechanism in the user form data validation mechanisms for guaranteeing the freshness of the submitted data. This unique anti-CSRF token is the suggested standard for all secure J2EE applications and should exist in all web application elements that can affect business logic (Alur, p. 77, 2013). An attacker could impersonate a legitimate user of the web application by exploiting this vulnerability with the help of the aforementioned HTTP-only flag-enabled cookies (OWASP, 2014f).

Results
SQL Injection: Burp Suite was the only scanner to detect the SQL Injection vulnerabilities by appending an encoded injection to the JSESSIONID cookie and inside the form submissions. It detected 4 of these cases.

Cross-Site Scripting: None of the scanners were able to detect this vulnerability. This vulnerability was only discovered by the researcher after inspecting the code line-by-line for instances of any code missing proper data validation mechanisms. Indeed, there is only so much that an automated black-box vulnerability scanner can do to discover vulnerabilities. As Suto mentioned – these dynamic scanners are meant to discover the "low hanging fruit" (Suto, p. 13, 2010.) White-box code inspection, manual black-box testing, and architecture analysis must all play a part in discovering these vulnerabilities.

Cleartext Credentials: This was peculiar. Acunetix was not able to detect an instance of broken authentication in this case, even when the HTTP Sniffer function (separate from the

overall scanner module) was configured to look for this vulnerability. Burp Suite and IRONWASP were able to detect all 19 instances of this vulnerability.

Insecure Cookies: Burp Suite outperformed all scanners in this case, producing 20 unique cases; Acunetix discovered only one instance, and none of the other scanners were able to detect the insecure cookies.

Session token in URL: Surprisingly, Acunetix, ZAP, and Arachni did not find this vulnerability that was so easy to spot with the human eye. Burp Suite and IRONWASP were able to see that the JESSESSIONID was transmitted in the URL bar and found all 3 instances.

Password auto-enabled: This simple security misconfiguration was detected in full by Burp Suite and ZAP – 12 instances; Acunetix and Arachni discovered 8 apiece and IRONWASP found none.

Missing anti-Cross-Site Request Forgery token: Acunetix, ZAP, and Arachni all tied for maximum detection on this vulnerability, finding 8 instances of HTML forms without Cross-Site Request Forgery protection. Burp Suite discovered one instance, and IRONWASP found none.

Java EE 7 applications rely primarily on security policy files for configuring authentication, authorization, and encryption. Java EE was designed with the intention of "shielding application developers from the complexity of implementing security features" (Jendrock, p. 48, 2014). It also "provides standard login mechanisms so that application developers do not have to implement these mechanisms in their applications" (Jendrock, p. 48, 2014). Security restraints must be defined to specify authorization and authentication controls. Duke's Forest was implemented with default security settings provided by Java EE 7 mechanisms, which ignores the 5th item on the OWASP Top 10 list of 2013 Vulnerabilities – Security Misconfiguration (OWASP, 2014). The methods provided by Java EE might not be sufficient for each individual web application. For instance, data validation for ZIP codes should be different than forum-like text boxes that are meant to accept HTML input. In this case, dynamic scanners are helpful to developers that might not have extensive experience implementing the data validation methods provided by Java EE for custom applications.

The vulnerability detection rate of these recommended web vulnerability scanners on other realistic web applications may be different, depending on the application. This is apparent when comparing the results of the web vulnerability scanners in the context of Duke's Forest, and Chen's WAVSEP test cases. However, the scan results on the Duke's Forest application signify that open source scanners not only present a cost-effective option to developers and testers – they are a viable option instead of proprietary scanners in the cases mentioned above.

In any software development company, the cost of these scanners must be considered along with the detection accuracy. Burp Suite offers a low-cost solution to companies that are more unwilling to purchase the expensive software license that proprietary scanners require for one computer. Acunetix's licensing options range between approximately $3,000-12,000 per year. HP WebInspect offers rates starting at $1,500 for one IP address and one computer, with rising licensing costs thereafter. IBM Appscan's pricing ranges between approximately $10,000-38,000 per year (Chen, 2014).

Dynamic web vulnerability scanners should never be the only solution for discovering software security flaws, but using open source web vulnerability scanners earlier in the software development lifecycle will increase early detection rates, lower security assessment workloads by using automated tools, and decrease total cost over the product's lifecycle by limiting expensive licensing costs.

6. CONCLUSION

Dynamic web vulnerability scanners should never be the only solution for discovering software security flaws, but using open source web vulnerability scanners earlier in the software development lifecycle will increase early detection rates, lower security assessment workloads performed before application deployment, and decrease total cost over the product's lifecycle by limiting expensive licensing costs.

This paper presented a low-cost alternative based on open source tools to high-cost

proprietary black-box web vulnerability scanners and supported this alternative combination of tools with the results of scans on the Duke's Forest application and scans performed by Shay Chen's WAVSEP yearly benchmark. The results of this paper's evaluation clearly show:

- The detection accuracy with these tools is more accurate than the detection accuracy with proprietary web vulnerability scanners in the test case provided by this evaluation.
- The input vector and attack vector support from these scanners can cover nearly every area of support by proprietary web vulnerability scanners

We also hope that future research and development will create an aggregate tool for integrating select functions from these recommended web vulnerability scanners using the APIs provided by the application developers. An aggregate tool utilizing the strongest capabilities of these open source products will create a web vulnerability scanner that is truly more powerful than the sum of its parts.

7. ACKNOWLEDGEMENTS

The research for this paper was financially supported by the National Science Foundation, under grant no. 1241440. In developing the ideas presented here, I have received helpful input from Neil Bahadur, Ashley Corbett, and Lavelle Perry. I also thank my three faculty mentors – Dr. Diane Murphy, Dr. Ali Bicak, and Dr. Michelle Liu – for guiding me in this process.

8. REFERENCES

Bennetts, S. (2014, May 21). Zed Attack Proxy (Version 2.3.1) [Computer software]. Retrieved July 1, 2014, from https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

Cenzic, Inc. (2014). Cenzic Application Vulnerability Trends Report [White Paper].

Chen, S. (2014). Security Tools Benchmarking: WAVSEP Web Application Scanner Benchmark 2014.

Dawson, I. (2012). Broken Logic: Avoiding the Test Site Fallacy. [White paper].

Doupe, A., Cova, M., & Vigna, G. (2010). Why Johnny Can't Pentest: An Analysis of Black-box Web Vulnerability Scanners. 7th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, 1-21. Retrieved June 31, 2014.

Graff, M., & van Wyk, K.R. (2003). Secure coding: Principles and practices. San Francisco, CA: O'Reilly.

Halfond, W. G., Choudhary, S. R., & Orso, A. (2011). Improving Penetration Testing through Static and Dynamic Analysis. Software Testing, Verification and Reliability. doi: 10.1002/stvr.450

Kernighan, B. W., & Pike, R. (1999). The practice of programming. Reading, MA: Addison-Wesley.

Kuppan, L. (2014, April 15). IRONWASP (Version 0.9.8.4) [Computer software]. Retrieved September 15, 2014, from https://ironwasp.org/about.html

Laskos, T. (2014, September 7). Arachni Web Application Security Scanner Framework (Version 1.0.1) [Computer software]. Retrieved September 20, 2014, from http://www.arachni-scanner.com/

Jendrock, E., Cervera-Navarro, R., Evans, I., Haase, K., Markito, W., & Oracle Corp. (2014, January 1). Java EE 7 Tutorial.

MacDonald, N., & Feiman, J. (2014). *Magic Quadrant for Application Security Testing* (Rep.). Stamford, CT: Gartner.

Malks, D., & Crupi, J. (2013). Synchronizer or Deja vu Token. In D. Alur (Author), Core J2EE Patterns: Best Practices and Design Strategies (pp. 77-87). Upper Saddle River, NJ: Prentice Hall.

OWASP Foundation. (2009). Code Review Guide (Vol. 1.1). Bel Air, MD: OWASP Foundation.

OWASP Foundation. (2014a). A1 - Injection. Retrieved September 3, 2014, from https://www.owasp.org/index.php/Top_10_2013-A1-Injection

OWASP Foundation. (2014b). A2 – Broken Authentication and Session Management. Retrieved September 3, 2014, from https://www.owasp.org/index.php/Top_10_2013- A2-Broken_Authentication_and_Session_Management

OWASP Foundation. (2014c). A3 – Cross Site Scripting. Retrieved September 3, 2014, from https://www.owasp.org/index.php/Top_10_2013-A3-Cross-Site_Scripting_(XSS)

OWASP Foundation. (2014d). A6 – Sensitive Data Exposure. Retrieved September 3, 2014, from https://www.owasp.org/index.php/Top_10_2013-A6-Sensitive_Data_Exposure

OWASP Foundation. (2014e). A7 – Missing Function Level Access Control. Retrieved September 3, 2014, from https://www.owasp.org/index.php/Top_10_2013-A7-Missing_Function_Level_Access_Control

OWASP Foundation. (2014f). A8 – Cross Site Request Forgery. Retrieved September 3, 2014, from https://www.owasp.org/index.php/Top_10_2013-A8-Cross-Site_Request_Forgery_(CSRF)

OWASP Foundation. (2014g). WebGoat Project (Version 6.0) [Computer software]. Retrieved September 24, 2014, from https://www.owasp.org/index.php/Category%3AOWASP_WebGoat_Project

Open Web Application Security Project. (2013). OWASP Top Ten Project. Retrieved from https://www.owasp.org/index.php/Category%3AOWASP_Top_Ten_Project

Portswigger Web Security. (2014, July 22). Burp Suite Professional (Version 1.6.03) [Computer software].

Stevens, J., & Cigital. (2011). When All You Have is a Hammer. [White paper].

Suto, L. (2010). Analyzing the Accuracy and Time Costs of Web Application Security Scanners. *Analyzing the Accuracy and Time Costs of Web Application Security Scanners,* 1-20. [White paper].

Tung, Y., Tseng, S., Shih, J., & Shan, S. (2013). *A cost-effective approach to evaluating security vulnerability scanner*. Paper presented at the Network Operations and Management Symposium (APNOMS), 2013 15th Asia-Pacific.

Web Application Security Scanner Evaluation Criteria. (2009). *Web Application Security Consortium,* 1-26.

APPENDIX 1

Duke's Forest Scan Results

Key:

- Burp Suite (low cost)
- Open Source
- Highest Detection Count
- False Positive Vulnerabilities

	Vulnerability	Acunetix	Burp Suite	IRONWASP	ZAP	W3AF	Arachni
1	Application Error Message	13					
2	HTML Form w/o CSRF Protection	8	1		8		8
3	User Credentials in Clear text	3	***19***	***19***			
4	Clickjacking	1	14	14	14		14
5	OPTIONS method is enabled	1					
6	Possible sensitive files	22					
7	Session cookie without secure flag reset	1	20				
8	Content type is not specified	7					
9	GHDB: Outlook PST File	1					
10	GHDB: Outlook Postscript File	1					
11	Password input: auto-complete enabled	8	12		12		8
12	SQL Injection	0	4				
13	Session token in URL		3	3			
14	Cross Domain Referrer Leakage		1	1			
15	Email Addresses disclosed			1			
16	Web Browser XSS Protection not enabled			1			

APPENDIX 2

Key:
= supported by Burp Suite
= supported by recommended open source scanner
= supported by proprietary
= not supported

	Appscan	WebInspect	Acunetix	w3af	Burp	NTOSpider	Arachni	ZAP	IRONWASP
#	1	2	3	4	5	6	7	8	9
logo									
COUNT	30	29	25	23	19	**19**	17	17	17
SQLi	✓	✓	✓	✓	✓	✓	✓	✓	✓
BSQLi	✓	✓	✓	✗	✗	✗	✓	✓	✓
SSJSi	✗	✓	✓	✓	✓	✓	✗	✗	✗
RXSS	✓	✓	✓	✓	✓	✓	✓	✓	✓
PXSS	✓	✓	✓	✓	✗	✓	✗	✓	✓
DXXX	✓	✓	✓	✗	✗	✗	✓	✗	✓
JSONh	✓	✓	✗	✓	✓	✗	✗	✗	✗
LFI	✓	✓	✓	✓	✓	✓	✓	✓	✓
RFI	✓	✓	✓	✓	✓	✓	✓	✓	✓
CMDExec	✓	✓	✓	✓	✓	✓	✓	✓	✓
UPLOAD	✓	✓	✓	✓	✓	✓	✓	✗	✗
REDIRECT	✓	✓	✓	✓	✓	✓	✓	✓	✓
CRLFI	✓	✓	✓	✓	✓	✓	✓	✓	✓
LDAPI	✓	✓	✓	✓	✓	✗	✓	✓	✓
XPATHi	✓	✓	✓	✓	✓	✗	✓	✓	✓
Mxi	✓	✗	✓	✓	✗	✗	✗	✗	✗
SSI	✓	✓	✗	✓	✗	✗	✗	✓	✓
FORMATi	✓	✓	✗	✓	✗	✗	✗	✗	✗
CODEi	✓	✗	✓	✗	✗	✗	✓	✓	✓
XMLi	✓	✗	✗	✗	✓	✗	✗	✗	✗
Eli	✗	✓	✓	✗	✗	✗	✗	✗	✗
BUFFERo	✓	✓	✗	✓	✗	✓	✗	✗	✗
INTEGERo	✓	✗	✗	✗	✗	✗	✗	✗	✗
CODEDisc	✓	✓	✓	✓	✓	✓	✓	✗	✗
BACKUPf	✓	✓	✓	✓	✓	✓	✓	✓	✗
PADDING	✗	✓	✓	✗	✗	✗	✗	✗	✗
AUTHb	✓	✓	✓	✓	✗	✓	✗	✓	✗
PRIVe	✓	✓	✗	✗	✓	✓	✗	✗	✗
XXE	✓	✓	✓	✗	✓	✗	✗	✗	✗
SESSION	✓	✓	✗	✓	✓	✓	✗	✓	✓
FIXATION	✓	✓	✓	✗	✗	✓	✓	✓	✓
CSRF	✓	✓	✓	✗	✗	✓	✓	✓	✗
ADoS	✓	✓	✓	✓	✓	✗	✗	✗	✗

Attack Vector Support. Adapted from WAVSEP.

ISSN: 2167-1508
v7 n3319

APPENDIX 3

Key:
= supported by Burp Suite
= supported by recommended open source scanner
= supported by proprietary
= not supported

	Burp	Appscan	NTOSpider	IRONWASP	WebInspect	ZAP	w3af	Arachni
#	1	2	3	4	5	6	7	8
logo								
COUNT	19	17	16	13	13	11	8	7
GET	✓	✓	✓	✓	✓	✓	✓	✓
POST	✓	✓	✓	✓	✓	✓	✓	✓
COOKIE	✓	✓	✓	✓	✓	✓	✓	✓
HEADER	✓	✓	✓	✓	✓	✓	✓	✓
SECRET	✓	✓	✓	✓	✓	✗	✗	✓
Pname	✓	✓	✗	✓	✗	✗	✓	✓
XML	✓	✓	✓	✓	✓	✓	✗	✗
XmlATT	✓	✓	✓	✓	✓	✓	✗	✗
XmlTAG	✓	✗	✗	✓	✗	✗	✗	✗
JSON	✓	✓	✓	✓	✓	✓	✗	✗
.NetENC	✗	✗	✗	✗	✗	✗	✗	✗
AMF	✓	✓	✓	✗	✓	✗	✗	✗
JavaSER	✗	✗	✗	✗	✗	✗	✗	✗
.NetSER	✗	✗	✗	✗	✗	✗	✗	✗
WCF	✗	✓	✓	✗	✓	✗	✗	✗
WCF-Bin	✗	✗	✗	✗	✗	✗	✗	✗
WebSock	✗	✗	✗	✗	✗	✗	✗	✗
DWR	✗	✗	✓	✗	✗	✗	✗	✗
Custom	✓	✓	✗	✓	✗	✗	✗	✗
DIR	✓	✓	✓	✗	✗	✗	✓	✗
FILE	✓	✓	✓	✗	✗	✗	✓	✗
Path	✓	✓	✓	✓	✗	✗	✓	✓
NetXml	✓	✗	✗	✓	✓	✗	✗	✗
NestJSON	✓	✓	✗	✗	✓	✗	✗	✗
JsonPName	✓	✗	✗	✗	✗	✗	✗	✗
Multipart	✓	✓	✓	✓	✓	✓	✗	✗
GWT	✗	✗	✓	✗	✗	✓	✗	✗
ODataID	✗	✗	✗	✗	✗	✓	✗	✗
ODataFilt	✗	✗	✗	✗	✗	✓	✗	✗

Input Vector Support. Adapted from WAVSEP.