Blog Final, Joseph Avanzato, Ashutosh Bhave

Introduction & Methodology

Web-Applications are typically sophisticated, feature-rich and complex pieces of public-facing software designed to handle interaction between remote clients and some form of data store or internally hosted component nested on a web-server. As such, ensuring the security of these interactions in regards to confidentiality, authentication and integrity of transactions is of paramount importance for web-developers but often security is seen as second to features in the development pipeline. Typical security issues for web-applications will include SQL Injection, Code Execution, Cross Site Scripting, File Inclusion and Cross-Site Request Forgery attempts (WASC, 2009). Sometimes these vulnerabilities are obvious and easily exploitable but often they are more nuanced and require particular attention paid to minute details either in the internal logic of a web-application or perhaps identifying some known flaw which has presented itself in a certain response from the web-application.

Identifying these vulnerabilities can be a time-consuming and arduous process for the average developer or penetration tester and as such many utilities have sprung into existence seeking to automate or otherwise assist security researchers in understanding how a specific web-application is vulnerable and how these potential exploit avenues can be mitigated or prevented in future updates. As such, understanding the strengths and weaknesses of individual application vulnerability scanners is important when deciding which application may suit your particular testing requirements. While there are many similarities among application scanners, no two scanners are created equal and this work seeks to explore the differences which may exist between various open-source utilities such as *ZAP*, *Vega*, *Arachni* and *W3af*. Gathering these metrics will be performed via the scanning of known-vulnerable web-applications designed for testing purposes; in particular, the OWASP BWA project was installed on top of a standard Linux/Apache/PHP/MySQL (*LAMP)* stack resting within an Ubuntu virtual machine. Specifically, we tested using the web-applications known as *Mutillidae II*, *Bricks* and *ZAPWAVE* which are all contained within the OWASP BWA project.

The previously mentioned web application suite provides security researchers with a fully-faceted test-bed, enabling the usage of either automated vulnerability scanners or manual exploitation in a safe and documented environment in order to understand inherent weaknesses in poorly designed or executed web-applications. Testing vulnerability scanners on these sorts of applications with set amounts of known exploits can help to assess the accuracy or performance of an individual scanner with respect to similar software. The types of utilities in use for this work included mainly open-source vulnerability scanners such as *ZAP*, *Nikto, Vega, Arachni, Watobo, Wapiti* and *W3af*; all of the listed utilities are available free and open-source in contrast to commercial level applications such as *Nessus* or *Nexpose* (Bau, Bursztein, Gupta, et al; 2010; Fonseca, Vieira, Madeira, 2007). While they may not provide the same complete set of features as an enterprise-scale implementation, these types of free utilities are in usage across the world by amateurs and professionals alike seeking to test the security of developed or deployed web-applications. As such, evaluating their accuracy with respect to one another in regards to vulnerability detection rates can give some insight to researchers deciding which application to utilize for their specific needs.

Being able to assess the state of a web-application with regards to potential vulnerabilities is important in identifying where these weaknesses exist, how prevalent they are and potential mitigation techniques which may be available in dealing with the associated risks. Understanding where and how different application scanners may fail to pick up on certain risks as well as developing custom scripts for stored XSS vulnerability detection will help to outline both the accuracy rates for various utilities as well as demonstrating how security researchers can test for these types of vulnerabilities in their own applications. Black-Box testing for such an attack may be difficult and as such a more ‘partial-knowledge’/grey-hat solution may be required with an internal look into an applications configuration and data management techniques helping to shed light on any potential issues.

Results

Overall, we used ten separate web application scanners to test for efficiency ratings against three separate web-applications which were delivered via the OWASP Broken Web Application project. Application scans were performed and results were categorized with respect to major vulnerabilities such as XSS, SQLi, LFI/RFI, CMDi, Other (Passwords in plaintext, HTTPONLY header not set, etc) and Low-Priority Information Disclosures such as blank or unspecified character sets. The scanners used were also briefly examined with respect to a multitude of features, as shown in the chart below.

Scanner Name	**Cookie Inclusion**	**Authentication Support**	**Confidence Metrics**	**Vulnerability Prioritization**	**CLI**	**GUI**	**Open-source**	**Report Exporting**
Vega	Yes	Yes	No	Yes	No	Yes	Yes	No
Skipfish	Yes	Yes	No	Yes	Yes	No	Yes	Yes
Nikto	Yes	Yes	No	No	Yes	No	Yes	Yes
Watobo	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
Wapiti	Yes	Yes	No	No	Yes	No	Yes	Yes
Arachni	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Acunetix	Yes	Yes	No	Yes	No	Yes	No	Yes
ZAP	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
W3af	Yes	Yes	No	No	Yes	No	Yes	Yes
Netsparker	Yes	Yes	Yes	Yes	No	Yes	No	Yes

The above chart helps to give an overview for the scanners that were used with respect to their various functionality and usage types. The actual data results for each scanner against the three tested web-applications are included below and categorized within the previously given areas. This type of data helps to directly observe the similarities and differences in performance capabilities when observing a broad range of scanning utilities.

Scanner Name	**XSS**	**SQLi**	**CMDi**	**LFI/RFI**	**Other**	**Low Priority**	**Time Taken (Minutes)**
Vega	**5**	**6**	**7**	**2**	**39**	**135**	**55**
Skipfish	**17**	**2**	**2**	**9**	**72**	**262**	**36**
Nikto	**0**	**3**	**1**	**8**	**37**	**25**	**24**
Watobo	**16**	**6**	**0**	**2**	**21**	**65**	**5**
Wapiti	**23**	**15**	**0**	**0**	**0**	**0**	**210**
Arachni	**18**	**4**	**2**	**5**	**17**	**22**	**255**
Acunetix	**14**	**11**	**1**	**2**	**37**	**64**	**439**
ZAP	**25**	**23**	**1**	**7**	**64**	**362**	**155**
W3af	**3**	**0**	**2**	**1**	**8**	**5**	**4**
Netsparker	**6**	**1**	**1**	**1**	**26**	**195**	**185**

Scan Results against *Mutillidae II*

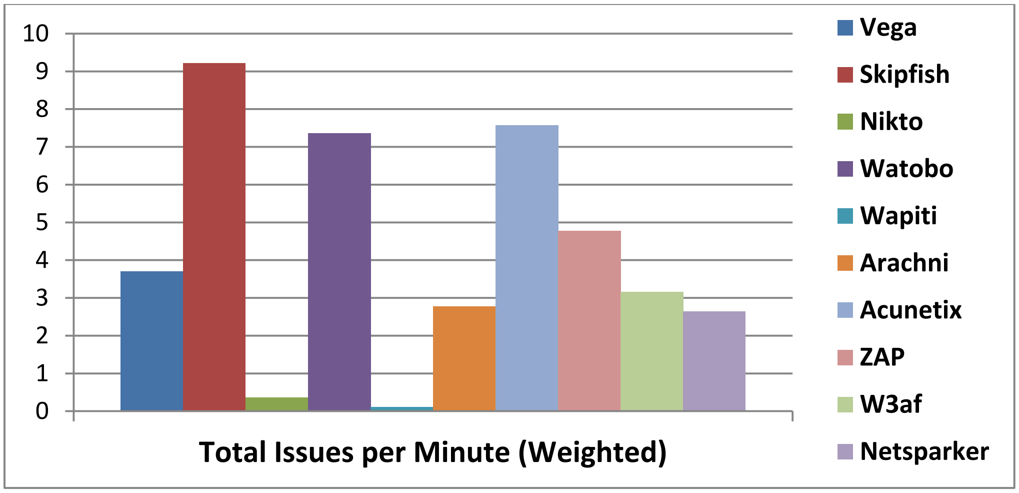
Scanner Name	XSS	SQLi	CMDi	LFI/RFI	Other	Low Priority	Time Taken (Minutes)
Vega	3	4	0	0	16	89	4
Skipfish	3	1	0	0	21	45	1
Nikto	0	0	0	0	3	7	2
Watobo	8	2	0	0	3	20	4
Wapiti	3	1	0	0	2	2	21
Arachni	5	2	0	0	6	34	6
Acunetix	3	2	0	0	5	18	3
ZAP	15	3	1	4	17	102	25
W3af	5	5	0	0	7	18	3
Netsparker	3	2	0	0	16	93	12

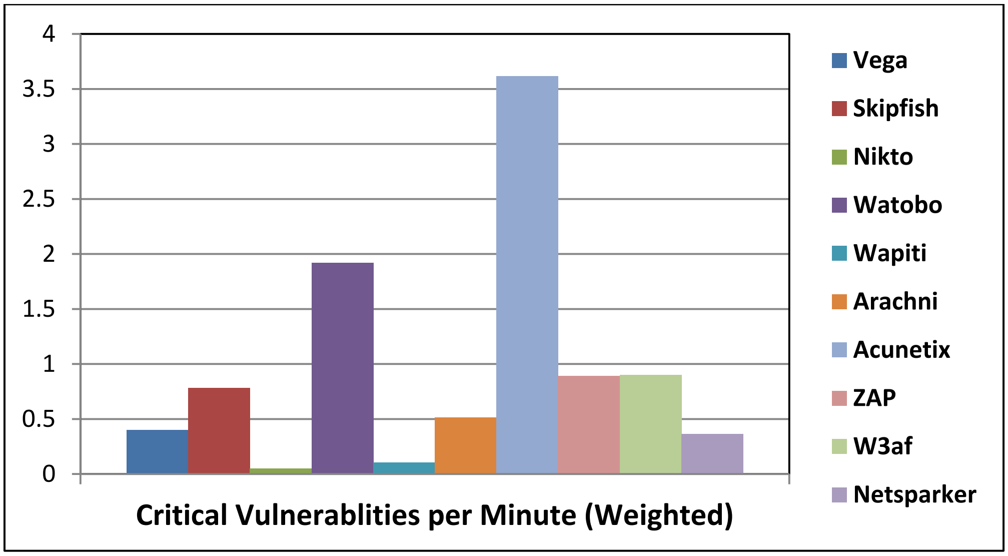
Scan Results against *ZAPWAVE*

Scanner Name	XSS	SQLi	CMDi	LFI/RFI	Other	Low Priority	Time Taken (Minutes)
Vega	12	19	0	1	13	36	32
Skipfish	23	2	1	12	32	248	20
Nikto	2	1	0	0	5	25	4
Watobo	12	14	0	2	24	45	5
Wapiti	13	13	0	0	0	0	74
Arachni	21	11	1	1	43	97	17
Acunetix	16	32	1	7	28	26	5
ZAP	15	43	0	2	54	125	43
W3af	2	1	4	7	17	19	3
Netsparker	12	27	4	4	28	192	37

Scan Results against *Bricks*

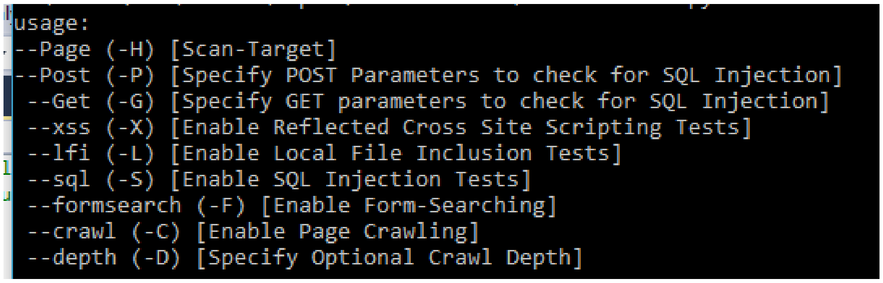
This data is interesting on its own but becomes more interesting when performing some basic analysis to try to understand how effective each scanner is when taking into account the time taken to perform a scan. To that end, we have performed a basic ‘total issues per minute’ and ‘critical vulnerabilities per minute’ metric computation which are weighted in accordance to the total amount of problems detected when compared to the scanner which detected the most issues. Charts representing these metrics for both cases are given below.





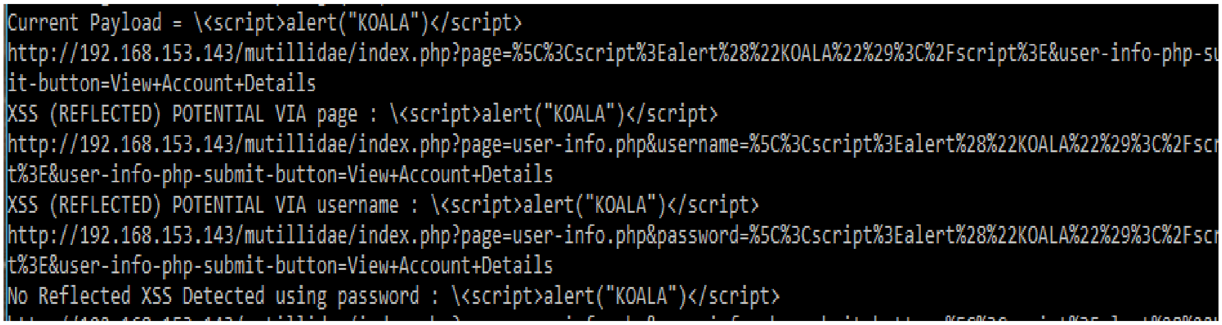
As is immediately evident from the above images, certain scanners seem to perform far better than others with respect to the two different categories of metric analysis. ***Skipfish*** seems to be the most effective at discovering 'total' issues which includes low-priority and other categorized problems while ***Acunetix*** and ***Watobo*** seem to be the most effective when examining purely critical vulnerabilities per minute rather than all issues combined. This type of analysis can help to give large enterprises the information necessary to make conclusions about which scanner to utilize when considering the fact that time is equal to money for corporations and saving time in terms of detecting potential vulnerabilities is important in order to reduce and manage overall risk while still bringing active scanning to efficient levels.

In addition to this work utilizing available vulnerability scanners, we also developed a custom utility designed to detect the presence of error-based MySQL injection, Reflected Cross Site Scripting (XSS) and Local File Inclusion (LFI). It also incorporates a basic crawling element which allows for the collection of embedded links on the provided page as well as the auto-scraping of HTML forms which are used to fuzz for SQLi/XSS in the associated modules. The developed scanner represents an extremely naïve initial attempt at understanding how to design a vulnerability scanner and we have begun to appreciate the developers of those which work well since we have learned it is extremely difficult to construct an efficient, accurate and well-performing application for this purpose. The scanner developed functions extremely inefficiently but does seem to pick up a subset of MySQL injections as well as perform extensive testing for Reflected XSS on scraped forms and LFI when provided a URI which includes the parameter upon which the user wishes to test. Some screenshots of the scanners operation are provided below.



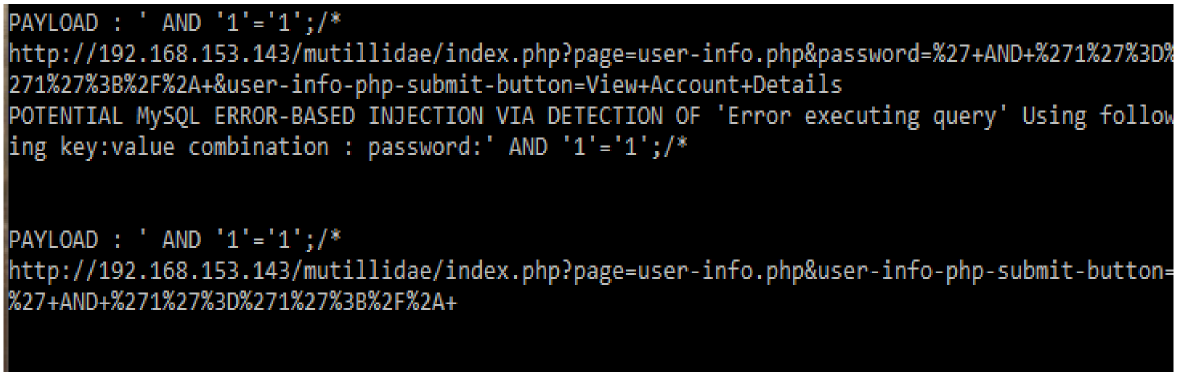
```
usage:
--Page (-H) [Scan-Target]
--Post (-P) [Specify POST Parameters to check for SQL Injection]
 --Get (-G) [Specify GET parameters to check for SQL Injection]
 --xss (-X) [Enable Reflected Cross Site Scripting Tests]
 --lfi (-L) [Enable Local File Inclusion Tests]
 --sql (-S) [Enable SQL Injection Tests]
 --formsearch (-F) [Enable Form-Searching]
 --crawl (-C) [Enable Page Crawling]
 --depth (-D) [Specify Optional Crawl Depth]
```

Basic Arguments used for Script Input



```
Current Payload = \<script>alert("KOALA")</script>
http://192.168.153.143/mutillidae/index.php?page=%5C%3Cscript%3Ealert%28%22KOALA%22%29%3C%2Fscript%3E&user-info-php-su
it-button=View+Account+Details
XSS (REFLECTED) POTENTIAL VIA page : \<script>alert("KOALA")</script>
http://192.168.153.143/mutillidae/index.php?page=user-info.php&username=%5C%3Cscript%3Ealert%28%22KOALA%22%29%3C%2Fscr
t%3E&user-info-php-submit-button=View+Account+Details
XSS (REFLECTED) POTENTIAL VIA username : \<script>alert("KOALA")</script>
http://192.168.153.143/mutillidae/index.php?page=user-info.php&password=%5C%3Cscript%3Ealert%28%22KOALA%22%29%3C%2Fscr
t%3E&user-info-php-submit-button=View+Account+Details
No Reflected XSS Detected using password : \<script>alert("KOALA")</script>
```

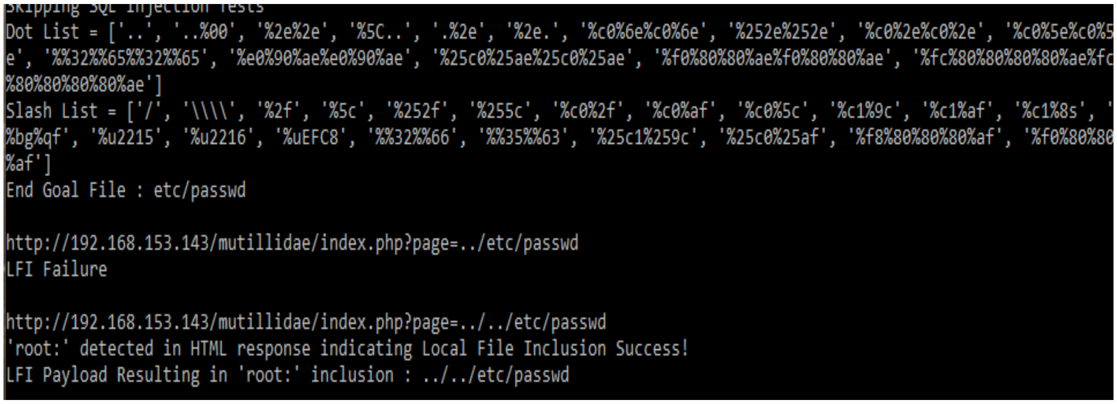
Example of Reflected XSS payload success and failure on different form parameters



```
PAYLOAD : ' AND '1'='1';/*
http://192.168.153.143/mutillidae/index.php?page=user-info.php&password=%27+AND+%271%27%3D%
271%27%3B%2F%2A+&user-info-php-submit-button=View+Account+Details
POTENTIAL MySQL ERROR-BASED INJECTION VIA DETECTION OF 'Error executing query' Using follow
ing key:value combination : password:' AND '1'='1';/*


PAYLOAD : ' AND '1'='1';/*
http://192.168.153.143/mutillidae/index.php?page=user-info.php&user-info-php-submit-button=
%27+AND+%271%27%3D%271%27%3B%2F%2A+
```

Example SQL Injection success and failure



```
Skipping SQL Injection Tests
Dot List = ['..', '..%00', '%2e%2e', '%5C..', '.%2e', '%2e.', '%c0%6e%c0%6e', '%252e%252e', '%c0%2e%c0%2e', '%c0%5e%c0%5
e', '%%32%%65%%32%%65', '%e0%90%ae%e0%90%ae', '%25c0%25ae%25c0%25ae', '%f0%80%80%ae%f0%80%80%ae', '%fc%80%80%80%80%ae%fc
%80%80%80%80%ae']
Slash List = ['/', '\\\\', '%2f', '%5c', '%252f', '%255c', '%c0%2f', '%c0%af', '%c0%5c', '%c1%9c', '%c1%af', '%c1%8s', '
%bg%qf', '%u2215', '%u2216', '%uEFC8', '%%32%%66', '%%35%%63', '%25c1%259c', '%25c0%25af', '%f8%80%80%80%af', '%f0%80%80
%af']
End Goal File : etc/passwd

http://192.168.153.143/mutillidae/index.php?page=../etc/passwd
LFI Failure

http://192.168.153.143/mutillidae/index.php?page=../../etc/passwd
'root:' detected in HTML response indicating Local File Inclusion Success!
LFI Payload Resulting in 'root:' inclusion : ../../etc/passwd
```

Example demonstrating initial LFI failure then success using different payloads

Upon the successful completion of an assigned scan, a report will be generated in the scripts directory listing the payloads and URI combinations that resulted in the success of Reflected XSS, SQL injection or Local File Inclusion. As stated before, this script is very early in development and does not reflect a complete program but rather a starting point for naïve vulnerability scanning. It utilizes very basic functionality for the crawling component and simple for loops to iterate through different payload parameters and combine them in a dynamic and iterating fashion.

Conclusions & Future Work

It seems from our data analysis that ***Skipfish, Acunetix*** and ***Watobo*** are the three most efficient scanners with respect to a metric attempting to derive ‘vulnerability per minute’ for time analysis purposes. Allowing enterprises to determine which scanner might be worth using via a ‘per minute’ rate is important because time spent is equivalent to money spent in businesses and using an inefficient scanner would be a waste of time, especially if it is not detecting issues which another more prevalent scanner may pick up on. Overall this work helped to demonstrate the differences between the scanners tested in a manner that is not often studied, their time-based efficiency. Since this metric has the capability to be vastly different depending upon the target web-application's ability to respond and the hosting systems processing capabilities, this metric may vary but should remain similar in proportions regardless since all tests were run from the same host platform.

Additionally, our work in exploring the development of a custom vulnerability scanner helped to demonstrate the difficulties associated with such a task. We learned how hard it can be to make efficiently performing and accurate algorithms for vulnerability detection and gained great respect for those developers who actively contribute and maintain projects such as OWASP ZAP. Fundamentally, vulnerability scanning is not 'difficult' but making a robust, dynamic, accurate and efficient scanner from scratch which can handle the myriad of website formats and configurations can be an extremely time-consuming and arduous task.

Potential future work in both of these areas could include a variety of directions. A more controlled and highly weighted statistical analysis for the vulnerability scanners could be conducted which aims to introduce additional factors such as vulnerability confirmations, thread control, request per second control or other similar weighting ideals might be interesting to obtain more accurate metrics. A better controlled host environment along with more tightly configured scan operations in order to make them as relatable as possible would also contribute to producing more effective and accurate metrics for overall comparison. Additionally, work towards making our custom scanner more robust and dynamic with respect to HTML form detection, crawling capabilities, allowed user input and more vulnerability detection capabilities would be interesting in order to pursue the creation of a better scanning engine.

References

Fonseca, J., Vieira, M., & Madeira, H. (2007, December). Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks. In *Dependable Computing, 2007. PRDC 2007. 13th Pacific Rim International Symposium on* (pp. 365-372). IEEE.

Halfond, W. G., Viegas, J., & Orso, A. (2006, March). A classification of SQL-injection attacks and countermeasures. In Proceedings of the IEEE International Symposium on Secure Software Engineering (Vol. 1, pp. 13-15). IEEE.

Bau, J., Bursztein, E., Gupta, D., & Mitchell, J. (2010, May). State of the art: Automated black-box web application vulnerability testing. In *Security and Privacy (SP), 2010 IEEE Symposium on* (pp. 332-345). IEEE.

Faghani, M. R., & Saidi, H. (2009, August). Social networks' XSS worms. In Computational Science and Engineering, 2009. CSE'09. International Conference on (Vol. 4, pp. 1137-1141). IEEE.

Web Application Security Consortium (WASC). (2009). Web application security scanner evaluation criteria. *Version*, *1*, 1-26.