2022 International Conference on Intelligent Data Science Technologies and Applications (IDSTA) | 978-1-6654-9960-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/IDSTA55301.2022.9923130

Practical web security testing: Evolution of web application modules and open source testing tools

Mohammed Ali Kunda, Izzat Alsmadi
Department of computing and cyber security
Texas A&M, San Antonio
San Antonio, USA
E-mail: mkunda@tamusa.edu, ialsmadi@tamusa.edu

***Abstract*—Web application security testing is vital for preventing any security flaws in the design of web applications. A major challenge in web security testing is the continuous change and evolution of web design tools and modules. As such, most open source tools may not be up to date with catching up with recent technologies. In this paper, we reported our effort and experience testing our recently developed website (https://mysmartsa.com/). We utilized and reported vulnerabilities from several open-source security testing tools. We also reported efforts to debug and fix those security issues throughout the development process.**

***Index Terms*—Security Testing and Automation, Web applications, Security , Security tools, OWASP ZAP, SoapUI, Penetration Testing tools**

I. Introduction

Web applications became a popular and useful tool in this internet era as businesses could easily reach their target audience regardless of the time and location for exchange of goods or services. This made communication easier between both parties, and made convenient for customers and businesses as it automates the whole process However, security is a significant concern. Web applications are target of attacks as they are available to the public all the time, and with a couple of scripts, attackers can gain access to sensitive information, causing serious security and liability concerns if the web application is not secure. As such, the security of web applications cannot be overlooked. Web application security threats such as SQL injection, and Cross-Site Scripting are well known and can easily be detected by security tools. However, with new technologies, software libraries, operating systems, and hardware changes, developing applications become complex and large. This can result in many new loopholes for security threats and issues. Robust security testing practices should be required for producing reliable and secure software applications.

II. Types of Testing

Table I provides a general classification of the security testing tools selected on the basis of efficiency, accuracy, different testing options available and paid or open source. From an environment or platform perspectives, security testing tools are available in command-line interface applications, Web applications, and desktop applications. Functional, security and performance testing are the three main testing goals we focused on.

TABLE I
Comprehensive analysis of the chosen security testing tools, [5]

Tools Name	Type	Functional	Security	Performance	Paid/free
Burp Suite	Desktop	Yes	Yes	Yes	Open-Source,Paid version
ImmuniWeb	Web	No	Yes	No	Open-Source,Paid version
Metasploit	Command Line	No	Yes	No	Open-Source
Netsparker	Desktop	No	Yes	No	Open-Source, Paid version
OWASP ZAP	Desktop	No	Yes	Yes	Open-Source
PractiTest	Desktop	Yes	No	No	Paid version
Proxy Sniffer	Command Line	No	No	yes	Open-source
SoapUI	Desktop	Yes	Yes	Yes	Open-Source,Paid version
Vega	Desktop	No	Yes	No	Open-Source
Wapiti	Command Line	Yes	Yes	No	Open-source

A. Security Testing Workflow

In security testing, there are numerous tasks and methods that could be used, however, it can be made simplified using the following steps:

- **Identify testing goals and scope**: First, determine what should be tested, taking into account the organization and the requirements of customers. It must be made clear which application, network system, or code will be put to the test, and a plan for how it will be carried out will be laid down.
- **Search for and Select most relevant suitable tools**: after identifying the testing parts, decide which tools will be well suited for the testing, taking into consideration the feature of the testing tools.
- **Perform cycles of security and vulnerability testing** Perform vulnerability scans and break down the tasks in terms of which one is more important, using the tools specified earlier. Check that the tool scans for vulnerabilities such as SQL injection, Cross-Site Scripting (XSS), Session Management, and file inclusion vulnerabilities .
- **Check and Validate testing results** Manually verify scanner results to see what matters in the context of the application and company by viewing the application through the eyes of a potential hacker .
- **Report results, document and repeat until reaching testing goals**: Organize your findings into a formal security assessment report. Because each company or security testing project has its own set of requirements, a sensible approach to web application security testing should include most, if not all, of these phases.

III. Goals and Approaches

Security testing and development in this project were accomplished in parallel. Such test-driven or early testing approaches can eliminate any vulnerabilities in the initial stages which is best practice to avoid any major issues in the future. Our developed website (https://mysmartsa.com/) can help San Antonio public transportation commuters navigate local and public transportation information,i.e. arrival time, number of seats available, current location, etc.
Open-source tools such as OWASP ZAP (stands for Open Web Application Security Project Zed Attack Proxy) and SoapUI are used in the security testing for our web application. These tools do not require extensive knowledge of the target website and usually just proving the URL can be enough. We will report in the next sections, our efforts using OWASP ZAP and SoapUI.

A. Summary Of Security Testing Using OWASP ZAP And SoapUI

1) OWASP ZAP: The first report generated from the ZAP detected the following 8 types of vulnerabilities. For details on those vulnerabilities, The following materials are available to readers and relate to those security testing tools, for ZAP alerts documentation, must visit: https://www.zaproxy.org/docs/alerts/

- CSP: Wildcard Directive, Risk: Medium, Count: 2
- X-Frame-Options Header Not Set, Risk: Medium, Count: 9
- Absence of Anti-CSRF tokens Risk: Low, Count: 2
- Cross Domain JavaScript Source File Inclusion, Risk: Low, Count: 10
- Information Disclosure – Debug Error Messages, Risk: Low, Count: 1
- Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s). Risk: Low, Count: 27
- X-Content-Type-Options Header Missing, Risk: Low, Count: 24
- Cookie No HttpOnly Flag, Risk: Low, Count: 1
- Information Disclosure - Suspicious Comments, Risk: Informational, Count: 7

2) SoapUI: The first report generated from the SoapUI detected 1 vulnerability. For details on those vulnerabilities, reader can find available resources related to those security testing tools

- HTTP method fuzzing, Count: 12
  colorlinks=false, pdfborder=0 0 0

B. Security Testing: Debugging and fixing security alerts

In this section, we report our effort to debug and fix the security issues mentioned previously. Typically, to be able to do that, we have to properly understand the alerts from the report. References can be a good start to understanding the alert and when it is typically triggered. This can provide security testers with the exact information that they will need to identify the issue and why it was triggered. We will examine the vulnerabilities of web applications created using the Express and later Django frameworks.

C. Dealing with OWASP ZAP Security Alerts

This section provides the possible solution/fix for the security bugs found in the web application in section III-A1 for OWASP ZAP.

- Content Security Policy (CSP) Header is Not Set: This alert got triggered because the content security policy header not set. which is a layer added for security which detects and prevents attacks like cross-site scripting and data injection attacks.So to setup the header, testers had to check with the framework documentation for CSP headers as there are many ways to configure it.One must set up your web server to return the Content-Security-Policy HTTP Header and provide its values in order to regulate what resources the browser is permitted to load for your page in order to resolve the Content Security Policy (CSP) Header Not Set.
  - The Syntax is:
    `Content-Security-Policy:policy-directive`
    where:
    `policy-directive`
    consists of: ¡directive¿ ¡value¿ with no internal punctuation.
    Example:
    `Content-Security-Policy: default-src 'self'`
    http://example.com;
    For a full list of possible directives and more examples please check
    https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy.

1) X-Frame-Options Header Not Set: This alert was triggered because the X-frame option header was not set and the web pages can be embedded within any other website with no restriction. This issue can be solved in different methods. In this case, we installed a package called Helmet.js. The package can help securing express apps by setting various HTTP headers. After complete installation and configuration of this package, this alert was resolved and X-frame option was configured properly.
2) X-Content-Type-Options Header Missing: Similar to the previous alert, in this alert X-content-type options header was missing or was not set properly. This issue was also fixed using Helmet.js package. It provides us with tools to set no-sniff option

- Code for Express framwork:
  - `app.use(helmet.nosniff())`
- Code for Django framwork in settings.py:
  - `SECURE_CONTENT_TYPE_NOSNIFF = True`

3) Absence of Anti-CSRF tokens: With any forms on the website, it's very important to have security in place to protect those forms from XSS attacks. The reason

this alert got triggered is that since there was no CSRF token validation. Packages like CSURF creates a middle-ware for CSRF token creation and validation. Using this package to create a middle-ware, the anti-csrf token worked and the anti-csrf token alert was solved.

4) Cross-Domain JavaScript Source File Inclusion: This alert was triggered because ZAP detects that there were external JavaScript files that were used in the project such as Bootstrap, Google API font, etc. This alert is triggered to notify the security tester to check if the sources are trust-able. After careful examination, all the JavaScript files and sources were verified to be genuine.

5) Information Disclosure – Debug Error Messages: Generally for documentation, developer teams create comments in the code which ZAP identified as debugging messages that can help the attacker. This alert was fixed by removing the comments so that ZAP doesn't identify it as a debugging message. Comments are important for developers in future releases to make code more readable and maintainable. However, from a security perspective, comments should not exist to give hints for attackers who can reverse engineer applications and retrieve original code.

6) Information Disclosure - Suspicious Comments: Similar to the previous one, this alert got triggered because some comments were too detailed and gave information about storing the data in the database. Website's code extraction is typically simple; comments can be viewed by anyone using inspect element or view source code.

7) Cookie No HttpOnly Flag: A cookie has been established without the HttpOnly flag, which allows JavaScript to access the cookie. To prevent this alert, Tester has enable the HttpOnly settings, as this website is built using Django, here is how it can be fixed using this line of code in settings.py : (" 'CSRF_COOKIE_HTTPONLY = True'. ").

8) Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s): This alert notifies the tester that while making HTTP requests, the server name is leaked which can be helpful for attackers. To fix this in express, we used this piece of code which can disable and hide the section where HTTP request shows the server.

- app.disable(x-powered-by')

D. Dealing with SOAPUI Security alerts

Referring to section III-A2, the following solution has been suggested in this section to address the vulnerability discovered as well as how it was initially triggered in soapUI.

- HTTP Method Fuzzing: The approach is to block HTTP methods, GET and POST as these two methods are defined in the API. The attacker should not be able to overwrite data on a server or get data that shouldn't be revealed to clients by unexpected HTTP methods

E. Penetration testing tools and usage

Thanks to the open-source community's contributions, there are numerous penetration tools publicly available. In this research paper we will evaluate three of those tools: Jok3r, SQLmap, Nikto as this tools provides the most of the testing functions which is need for the security testing like network infratructure and web black-box testing, SQL injections, http headers, Cross site scripting, etc.

F. Jok3r, [1]

Jok3r [1] is a Python-based CLI program which is designed to assist penetration testers with network infrastructure and web black-box security tests, [2]. The tests below summarized our testing activities and results:

- After installing the tool, create a database first that will contain all of the test results.
- Choose the different options which is suitable for your web application security testing. Check all the various features for testing using the -h option.
- Run the attack with custom scripts for a particular type of testing.
- To view the test results, open the database used to store scan results.
- Test scan results:
  - robots.txt is not found or empty.This text file indicates the need for search engine crawlers to know which URLs are accessible and which are blocked.
  - SSLv2 & SSLv3 not offered: These secure sockets layers, which have versions 2 and 3, maintain the safety and security of internet connections but they aren't provided.
  - Null ciphers, anonymous null cipher (no authentication), export ciphers are not offered.
  - Strict transport security is not offered: This is the layer in charge of making sure websites connect to HTTPS (This layer does not exist in the tested application).

G. SQLMap' [3]

SQLMap ([3]) is an open-source penetration testing tool that automates the process of finding and exploiting SQL injection vulnerabilities and controlling database servers, [4]. The tests below summarized our testing activities and results:

- Start the offensive test on the target website with the payload you wish to use to test the database for SQL injection vulnerabilities.
- Scan: If there is no secure layer in between to protect the database from SQL injection, SQLMap tools locate and display the information obtained from the database.
- Test results may include user name and password, table names from the database, column names from the database, create tables, update & retrieve information, delete users, delete tables, etc.

H. Nikto, [7]

Nikto ([7]) is a open source web application scanner. It runs thorough testing on web servers for a variety of things like potentially harmful files and applications, checks for out-of-date versions on more than 1250 servers, and version-specific issues on more than 270 servers, [6].
The tests below summarized our testing activities and results:

- Run the vulnerability scanner with specific test options on the web application.
- Scans results:
  - The anti-click-jacking X-frame-options header is not present, There is a potential danger of a click-jacking attack on the web application since the anti-click-jacking X-frame-options header is missing.
  - The X-content-type-options header is not set, the browser will be able to perform MIME type sniffing. The browser will determine what type of content is present and how to handle it when it receives a response from the server.
  - The X-XSS protection header is not defined. Any pages on this website may be vulnerable to an XSS attack because the server is not set up to return a 'X-XSS-Protection' header.

Here is a summary of ZAP and Soapui security alerts and our debugging efforts:

- **Content Security Policy (CSP) Header Not Set**: Setting the content security policy header by consulting the framework documentation for syntax and using the inspect tools is helpful since it enables debugging of resources that were omitted from the header and allows for their addition to the content security policy.
- **X-Frame-Options Header Not Set**: Depending on which framework is used for building the web application, tester should refer to framework documentation for syntax, the X-frame-option can set, in express framework, install helmet.js package and configure the X-frame header which will fix this issue and secure your web application
- **Absence of Anti-CSRF tokens**: Whenever we use forms in web application, its important to make it secure with tokenization, its simply validation the data was not changed. There are many was to generate anti-csrf token, you can generate token on server side and validate on client side but different frameworks have their unique way of validating forms with tokens.
- **Cross Domain JavaScript Source File Inclusion**: Make sure all the external javascripts are from credible sources or load those JavaScript file locally which means have javascript code in the project locally.
- **Information Disclosure – Debug Error Messages**: Remove any debugging message during https request which can help attacker to gain inside information.
- **Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)**: Hide the server name in the HTTP request which used for building web application,in express we can hide the server name on server side by using 'app.disable('x-powered-by')'
- **X-Content-Type-Options Header Missing**: X-content option can be set by enabling the nosniff mode,in framework require a line of code which set the value of nosniff to true.
- **Cookie No HttpOnly Flag**: Enable Httponly flag for all cookie, In Django, you can accomplish this via 'CSRF_COOKIE_HTTPONLY = True'.
- **Information Disclosure - Suspicious Comments**: Make sure to remove the comments in the code which provides important and critical information about storing the data, functionality, etc.
- **HTTP Method Fuzzing**: Configure https methods so that attacker should not able to overwrite data by sending unexpected HTTP methods

IV. Conclusion

In this paper, we reported our experience to test a recently developed website for security issues or vulnerabilities. As a show case, we reported in this paper, using two desktop and three command-line interface open source security testing tools. We also reported our efforts to debug and fix those security issues as well as comparison of open-source providing an overview of the kind of testing the tools can perform as well as the application type whether it is desktop, web or command-line interface interface and further demonstrated a five-step security testing work cycle. Lastly, we wanted to demonstrate in this paper what vulnerabilities can exist, why they were triggered, and how to fix the security issues to make web applications secure.

Acknowledgments

This material is based upon work supported by the National Science Foundation (NSF) under Grant No. 2131193. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Jérémy Brun-Nouvion. Jok3r - network web pentest automation framework, 2019.
[2] Jérémy Brun-Nouvion. Jok3r - network web pentest automation framework, 2019.
[3] Bernardo Damele and Miroslav Stampar. Sqlmap, 2022.
[4] Bernardo Damele and Miroslav Stampar. sqlmapproject/sqlmap, 2022.
[5] Martin Lněnička and Jan Capek. Classification and evaluation of cloud-based testing tools: The case study of web applications' security testing. *Acta Informatica Pragensia*, 7:40–57, 06 2018.
[6] Chris Sullo and David Lodge. Nikto, 2022.
[7] Chris Sullo and David Lodge. sqlmapproject/sqlmap, 2022.