

## International Journal of Computer Science and Mobile Computing



A Monthly Journal of Computer Science and Information Technology

**ISSN 2320-088X**

**IMPACT FACTOR: 6.017**

*IJCSMC, Vol. 7, Issue. 12, December 2018, pg.197 – 212*

# Investigation and Comparison of Web Application Vulnerabilities Test Tools

**Muhammet BAYKARA**

Software Engineering Department & Firat University, Turkey

[mbaykara@firat.edu.tr](mailto:mbaykara@firat.edu.tr)

---

**Abstract**— Today web-based systems are very popular. These systems may have some inherent security vulnerabilities due to the languages they use. It is very important to identify these vulnerabilities for the development of quality and secure web applications. There are many commercial and open source applications that detect security vulnerabilities on the websites application level. However, developers are curious about which tools detected security vulnerabilities, and their performance rates. In this study, it is aimed to analyse the frequently used web vulnerability test tools with sample scenarios and compare these tools.

**Keywords**— Web application (WebApp), Security Scanners, Web vulnerabilities, Web vulnerability scanners (WVSs), Web application scanner (WASs)

---

## I. INTRODUCTION

People meet many important needs on the Internet websites. The volume of e-commerce in the market is growing exponentially each year. Amazon is one of the leaders of e-commerce. Amazon selected as the 3rd most valuable brand in the world by Fortune. This is proof of how serious e-commerce has reached [1]. The size of the market to reach these dimensions makes e-commerce sites an attractive target for attackers.

Web Apps are difficult to secure because they are naturally open to the public, including malicious users. In addition, input to web apps comes from HTTP requests. This input is difficult to process correctly. Improper or missing input validation will result in security vulnerabilities in web apps. Network vulnerability scanners, network firewalls, and use of Secure Socket Layer (SSL) do not fully secure a website [2]. Gartner Group estimates that more than 70% of attacks on a company's website or web apps are in the application layer, not a network or system layer [3].

WASs help us to reduce the number of vulnerabilities in web apps. In short, WASs scan the pages of a web app and look for vulnerabilities by simulating attacks on the application. Web app browsers can find many security vulnerabilities and cannot provide evidence that a stand-alone application is safe. WASs are implemented in the final stages of the software development lifecycle. Security should be designed and structured. Various tools and best practices should be implemented throughout the development life [4].

## II. ANATOMY OF WEB APPS

The Web Application Security Consortium (WASC) [5] defines a web app as a "software application executed by a web server that responds to dynamic web page requests over HTTP".

A web app includes a script on a web server that interacts with databases or other resources of dynamic content. Using Internet infrastructure, web apps allow service providers and clients to share and redirect information in a platform-independent manner [6].

Technologies used to create web apps: PHP, Active Server Pages (ASP), Perl, Common Gateway Interface (CGI), Java Server Pages (JSP), JavaScript, ASP.NET, VBScript and so on. Some of the broad categories of web app technologies are communication protocols, formats, server-side and client-side scripting language, browser add-ons, and web server APIs.

A web app has an n-layered architecture. Typically, there is a client (web browser), a web server, an application server (or several application servers), and a database server. Figure 1 is a basic view of a web app. There can be a firewall between the client and the server.

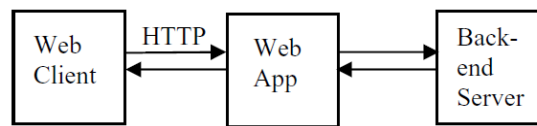


Figure 1. The working diagram of a typical web app

## III. THE ORIGIN OF WEB APP VULNERABILITIES

Web apps generally interact with the user through the FORM elements such as buttons, text boxes, etc. and the GET or POST variables. Improper processing of data items within HTTP requests causes the most critical security vulnerabilities in web apps. SSL does not prevent security vulnerabilities because it provides secure data transfer and does not examine HTTP requests.

Web apps are a waypoint for databases that hold critical application data and assets. Some of the main threats to the database server layer are SQL injection, unauthorized server access, and password cracking attacks. Most SQL injection vulnerability are caused by weak input validation. Most web apps store sensitive information in databases or in a file system. Developers often make mistakes in encryption techniques to protect this information. Because HTTP is a stateless protocol, web apps use separate mechanisms to maintain session state. A session is a series of interactions between the user and the web app during a single visit to the website. Generally, session management is performed using a unique string called Session ID that is transmitted to the web server with each request. Most web scripting languages support sessions through GET variables and/or cookies. If an attacker can guess or stole a session ID, he or she can change the session of another user.

## IV. WEB APP VULNERABILITIES AND SCANNERS

Web app scanner is an automated tool that analyse web apps for security vulnerabilities. In addition to searching for specific security vulnerabilities for web apps, tools also search for software coding errors, such as illegal input strings and buffer overflows.

The web app scanner searches an application by browsing web pages and performs a penetration test. More simply, it simulates the attacks of a web app and effectively analyzes it. This includes producing harmful inputs and then evaluating the response of the application. WAS performs different attack types. A handy attack, commonly known as fuzzing, sends random entries to the application in various sizes.

Penetration testing is a typical black box test approach. The limitation of this approach is that it does not examine the source code, so it is less likely to detect these vulnerabilities as a backdoor. However, it is very suitable for detecting input validation problems. In addition, client-side code (JavaScript, etc.) is available for the penetration tester and can provide important information about the internal operation of a web app.

This section discusses the vulnerabilities that a web app scanner should detect. These vulnerabilities will form the basis of an officially expressed requirement for mandatory features for a web app scanner. A useful classification of web security threats can be found in [7]. The Open Web Application Security Project (OWASP) publishes a list of the most critical web app vulnerabilities [8]. These and other efforts are included in Common Weakness Enumeration (CWE) [9].

Input validation weaknesses cause most web app vulnerabilities. Other weaknesses include the use of weak authentication mechanisms, logical weaknesses, unintentional disclosure of content and environmental information, and include low-level encoding weaknesses (such as buffer overflows). The combination of weak points often causes weakness.

Some common web-based attacks and vulnerabilities are given below:

- Cross Site Scripting (XSS) vulnerabilities; this vulnerability occurs when an attacker sends malicious data to a web app. Examples of such data include client-side scripts and hyperlinks to the site where an attacker is located. If the application collects data without proper verification and dynamically displays it on generated web pages, it displays harmful data in a legitimate user's browser. As a result, the attacker can process or steal the credentials of the legitimate user, impersonate the user, or execute malicious scripts on the user's machine.
- Injection vulnerabilities; this type of attack includes data injection, command injection, resource injection, and SQL injection attacks. SQL Injection does not properly filter out a web app user input and places it directly into a SQL statement. This may allow the data in the database to be explained and/or modified. Another possible object of injection is executable scripts that are forced to do things that the authors do not anticipate.
- Cookie poisoning is a technique used primarily to access the impersonation and privacy violation through manipulation of session cookies that protect the identity of the client. By doing these cookies, an attacker can imitate a valid client and thus gain information and perform actions on behalf of the victim.
- Invalid input; XSS, SQL Injection and cookie poisoning vulnerabilities are some of the specific examples of this problem. In addition, there are also some events such as dirty data and forms, improper use of hidden fields, overriding data in the array directory, the function call, the loop conditions in a format string, memory allocation, and the use of array distribution.
- Authorization, authentication, and access control vulnerabilities may allow the malicious user to take control of the application or back-end servers. This attack includes weak password management, the use of weak encryption methods, the use of elevation of privileges, insecure macro usage for dangerous functions, use of unwanted copies, authentication errors, and cryptographic errors.
- Incorrect error handling and reporting information can reveal information by opening doors to allow malicious users to predict sensitive information. This includes the catch NullPointerException, the empty catch block, and the extremely large "throws" declaration.

Some other security vulnerabilities:

- Denial of Service (DoS)
- Path Manipulation
- Broken Authentication
- Broken Access Control
- Sensitive Data Exposure
- Insecure Deserialization
- Security Misconfiguration
- Using Components with Known Vulnerabilities

## V. ANALYSIS OF TEST TOOLS

In this section, six commercial and open source test tools were examined. These are *Netsparker*, *Acunetix*, *Vega*, *OWASP ZAP*, *Wapiti* and *IronWASP*. To test each tool, a test was made on the <http://aspnet.testsparker.com/> address prepared by the Netsparker team.

### A. Netsparker

Netsparker is a web security testing tool. It analyzes and reports security vulnerabilities at the application level of any website. In addition, with using these vulnerabilities it can also extract data from the system, such as an attacker, or provide full access to the system. However, false positives are avoided. Netsparker provides full support for AJAX and JavaScript-based applications. Netsparker also provides full support for HTML5. Netsparker has two versions, desktop and cloud. The cloud version allows us to scan thousands of websites or web-based applications simultaneously [10].

Test Scenario;

The test was performed with version 4.8 of the Netsparker desktop app on Windows 10 (x64). The URL tested in the scenarios is the <http://aspnet.testsparker.com/> address.

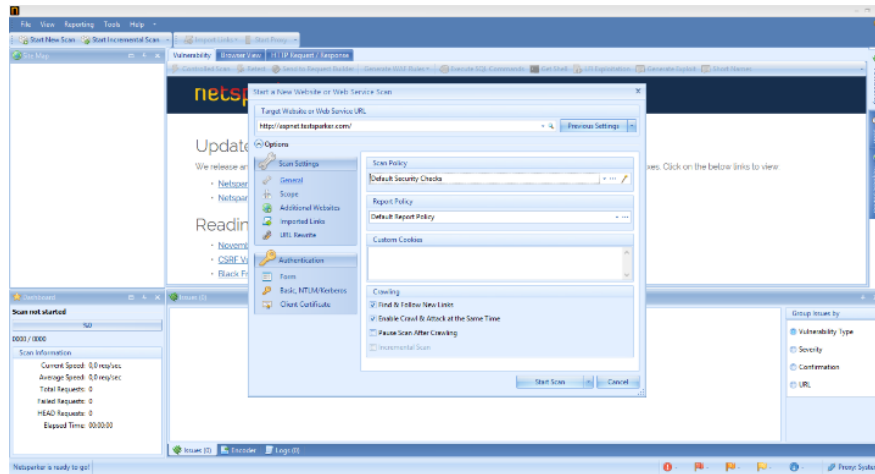


Figure 2. URL input and test configuration with Netsparker tool

When we start the program, we meet a very simple and useful interface that everyone can use (Figure 2). We're entering the URL of the site we want to scan to the *Target* section. In the *Scan Policy* section, we can select the configurations we make to scan the vulnerabilities that we set specifically. If a special reporting format has been prepared in the *Report Policy* section, we notify that it needs to report in that format. We'll continue here by default and click *Start Scan* to start scanning.

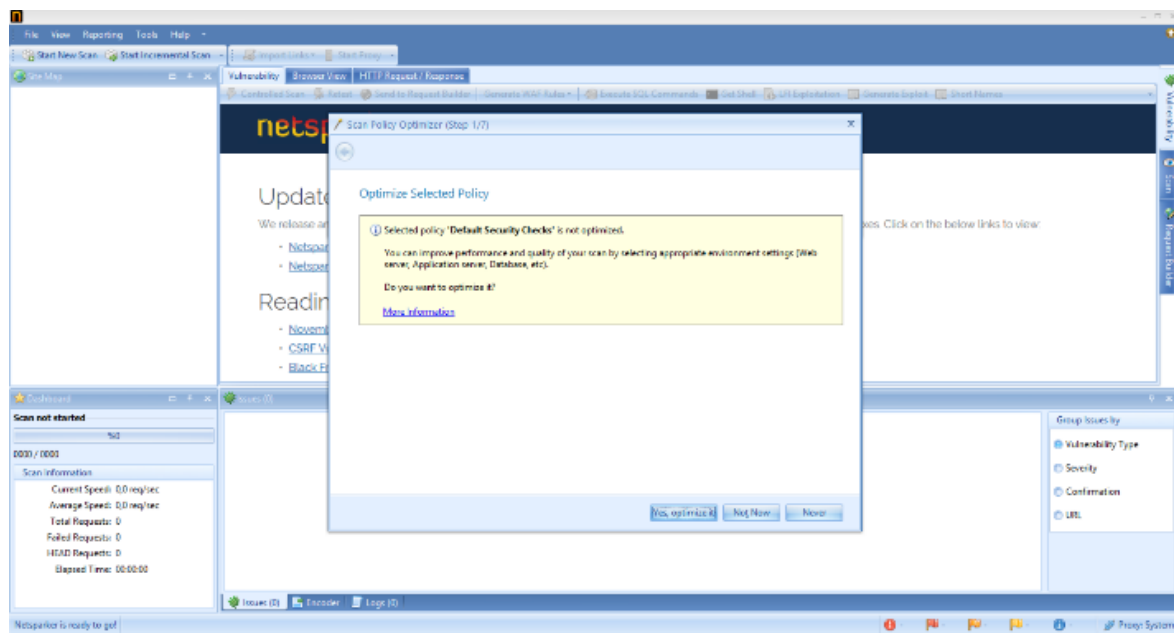


Figure 3. Optimize selected rules with Netsparker tool

In this case, the server optimization can be performed on the screen shown in Figure 3. If we do server optimization, we come across options such as operating system, web server, application server, database server and the desired arrangements are made. In this section, the default settings are selected and the scan is started.

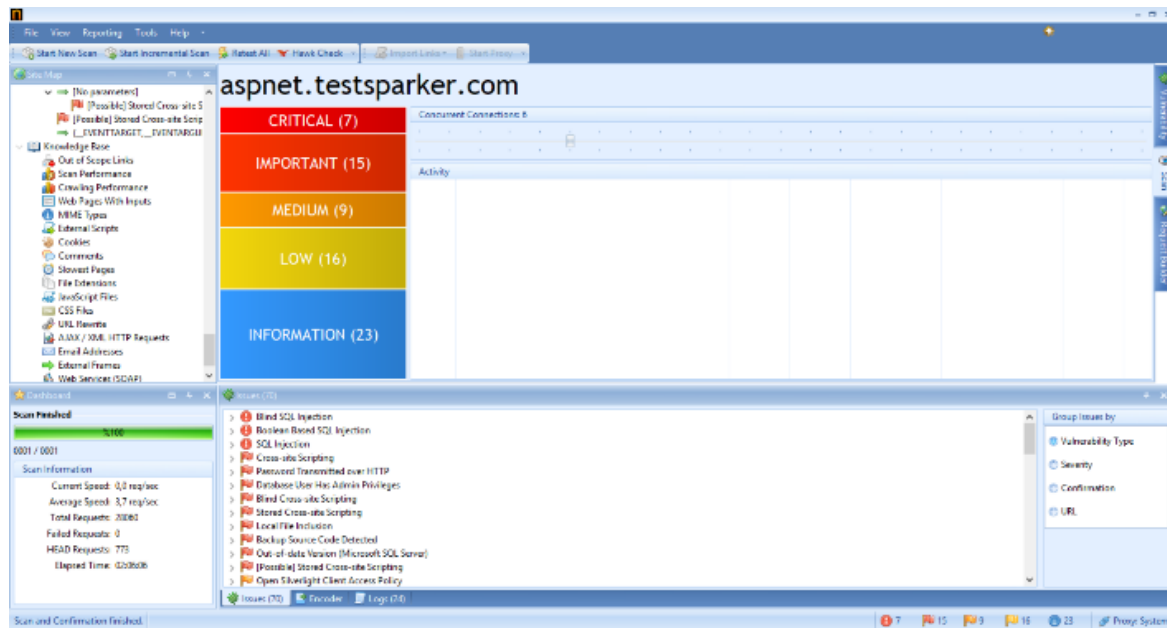


Figure 4. The results of scanning with Netsparker

Figure 4 shows the report/evaluation page when the scan is finished. The vulnerabilities found are listed in the in the bottom *issues* window according to their severity.

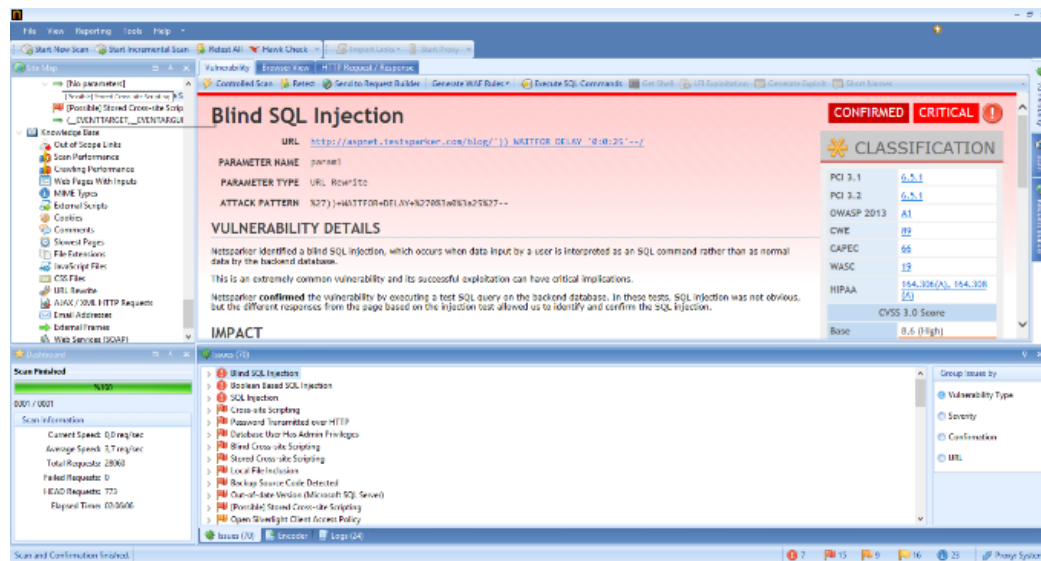


Figure 5. Detailed information about the listed vulnerabilities

To obtain detailed information about the vulnerabilities listed, simply click on the relevant vulnerabilities. Then in the above window a detailed information about the vulnerability is obtained. As it can be seen from Figure 5 there is a SQL Injection vulnerability in the system. If you want, click on the *Execute SQL Commands* button at the top of the system for take advantage of vulnerabilities as an attacker. Other options include other types of attacks, such as exploiting.

Scanning with Netsparker is comprehensive and has identified 47 vulnerabilities on the site. The scan took about 2 hours. The report for the vulnerabilities is very detailed and just as satisfactory. The disadvantage of such a comprehensive scan is that the scanning time is quite long. Netsparker is one of the best quality web security testing tools that can be used with user friendly interface, customizable searches, comprehensive scanning and reports. The biggest difference between Netsparker and its competitors is that it allows you to use the vulnerabilities as an attacker. The biggest disadvantage is that the license fee starts at \$ 5000 a year.

## B. Acunetix

Acunetix is a web security testing tool that thoroughly scans and controls especially HTML and JavaScript based websites. Software Development Lifecycle (SDLC) integrates with project management or bug tracking systems, includes comprehensive accordance reports. Acunetix detects and reports web app vulnerabilities. Acunetix is one of the most successful tools on the market to detect SQL Injection and XSS vulnerabilities. Acunetix also has the most advanced detection of WordPress vulnerabilities and a wide range of management and legal reports, including ISO 27001 and PCI accordance. Acunetix can limit false positives and easily access restricted areas. With the Acunetix v11 version, it has switched from the desktop application environment to the web-based user interface. In this way, it operates independently of the operating system via web browsers [11].

### Test Scenario:

The test was performed with Acunetix v11 on Windows 10 (x64).



Figure 6. Sign in for acunetix web scanner

When we start Acunetix, it direct to us to localhost. We login here. As shown in Figure 6, after entering the required information, an interface appears that seen in Figure 7.

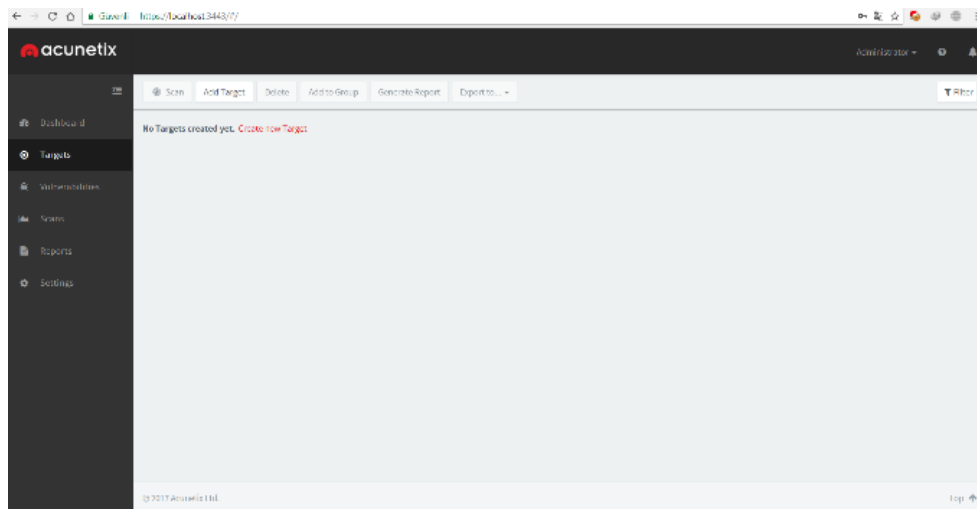


Figure 7. Acunetix web-based GUI

After logging in, it meets us user friendly and quite plain interface. We will first specify a new destination address to start the scan by clicking the *add target* button in the targets section.

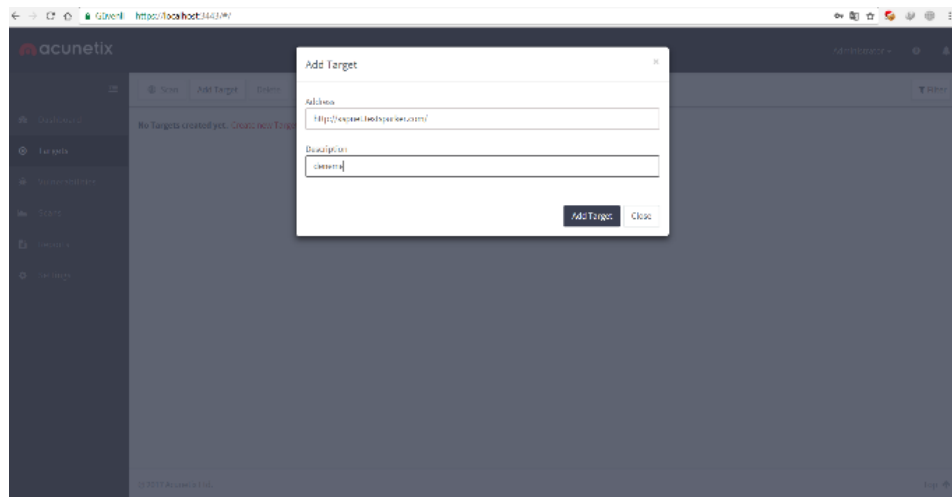


Figure 8. Entering the URL for testing with add target function

In the window that opens, we enter the URL of the destination site in the address field. In the *Description* section, we enter a sentence that defines the search we made. In this way, it is possible to follow up the test scenarios and system errors which are performed at different times of the tested web apps. Then, we add our target by clicking *add target* our target (Figure 8).

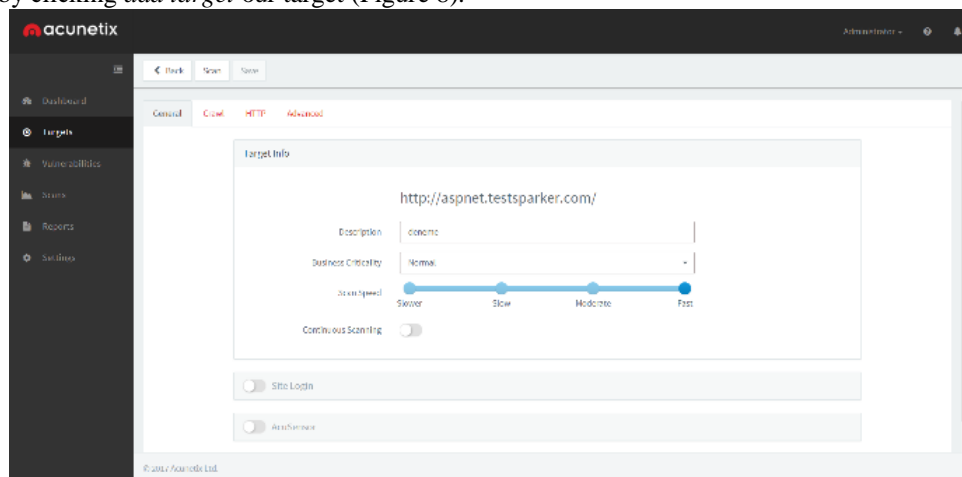


Figure 9. Settings for vulnerability scan (General Tab)

In this section given in Figure 9, you can do the customizations that we want before launching the scan. We can optionally adjust the speed of the scanning process, the severity level and many more options.

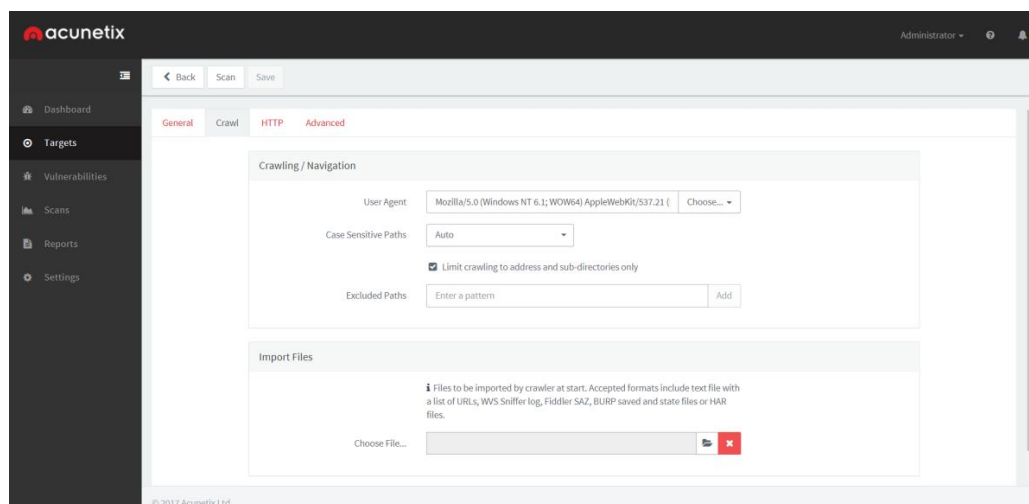


Figure 10. Settings for vulnerability scan (Crawl Tab)

After making the necessary adjustments, click the *Scan* button above.

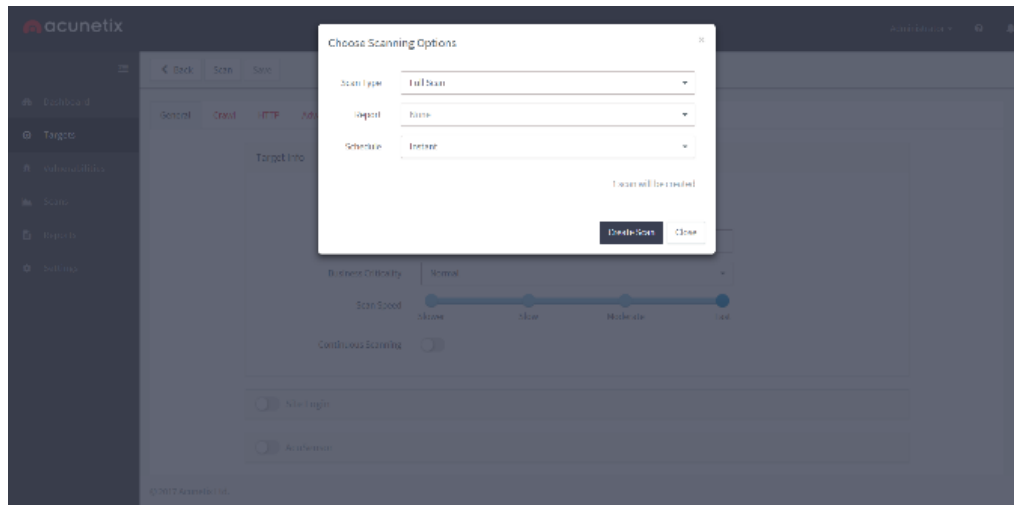


Figure 11. Selecting scanning type

As shown in Figure 11, there are three options in the pop-up browsing window. First of all, many scanning options such as full scan, high risk vulnerabilities, XSS are offered. In the second part, many other legal reporting options such as ISO 27001, NIST SP800 53, OWASP TOP 10 are offered for reporting. In the *Schedule* section, you can find options such as *Instant*, *Future Scan*. As shown in Figure 11, we continue with the fixed settings and click on the create scan button. So our configuration settings are now over and Acunetix has started scanning.

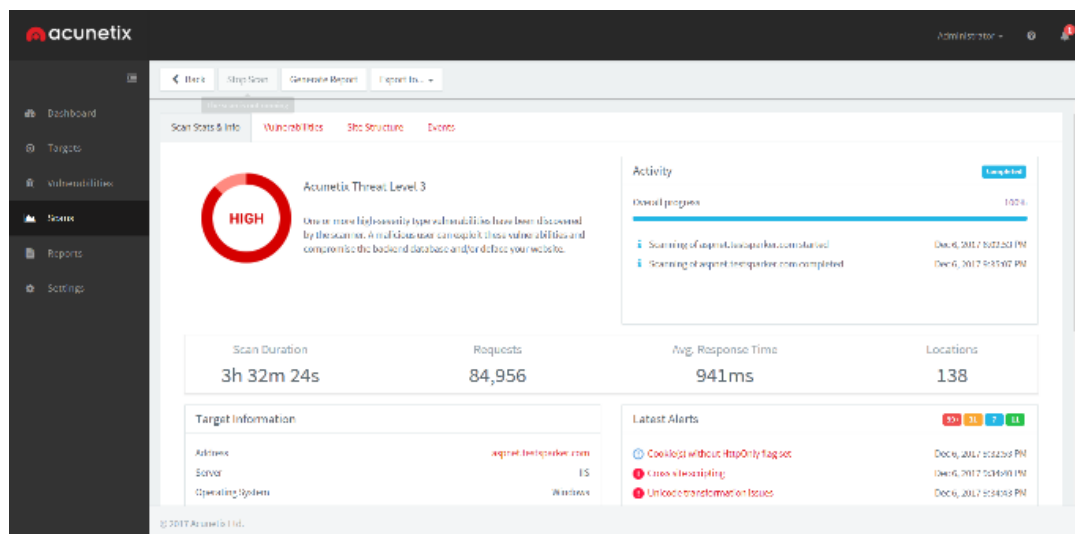


Figure 12. Checking detected vulnerabilities with acunetix scanner

After the scan process is completed, click on the *vulnerabilities* tab for detailed information about the detected vulnerabilities, and the found vulnerabilities are displayed in order of importance. For summary information, click on *Dashboard*. To generate a report, click on the *Generate Report* button. You can review the *Reports* section for reporting information.

Acunetix is ahead of its competitors with a very simple and user-friendly interface. Especially the richness of the reporting section provides a huge plus. In pre-scan configuration settings, it offers satisfactory options. In this study, as a result of the scan, 163 vulnerabilities have been detected on the website. With regard to the license fee, we see a much more reasonable price (starting at \$ 345) compared to other paid competitors by 2016 and 2017. In 2018, license prices start at \$ 4500 [11].



### C. Vega

Vega is a free and open source web security testing tool used to find vulnerabilities in web apps. The GUI of Vega is written in Java. Vega has two views. These are *Scanner* and *Proxy*. Vega interactive web app includes a blocker Proxy for debugging. Vega attack modules are written in JavaScript. Because of open source, these modules can be expanded with a JavaScript API and rewritten by the user [12].

#### Test Scenario:

The test was performed with Vega v1 on Windows 10 (x64).

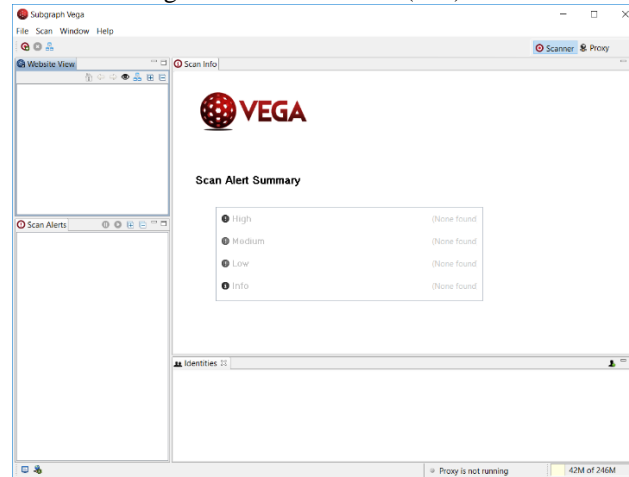


Figure 13. GUI of Vega web vulnerability scanner

When we start the Vega application as shown in Figure 13, it meets us with a simple interface. We first select the *Start New Scan* option by clicking the *Scan* tab above to start a new scan.

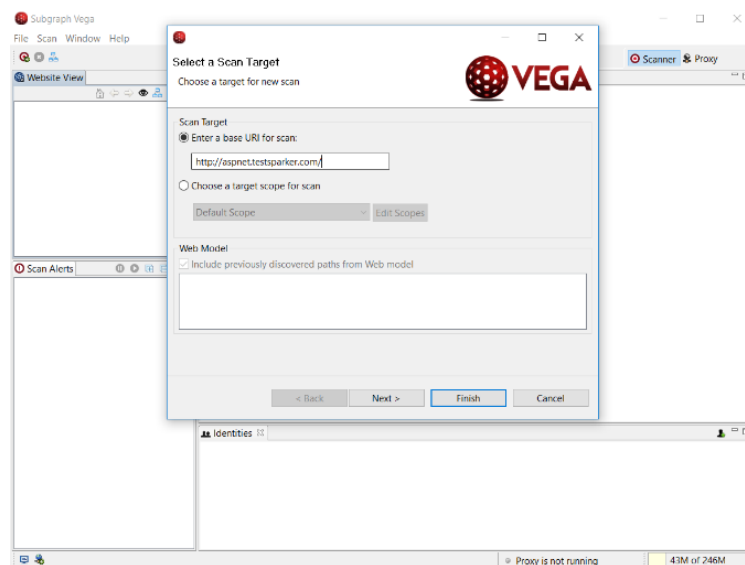


Figure 14. Entering the URL for scanning

In the interface that appears, we enter the URL of the destination site in the scan target. Then you can click Next button to configure the configuration settings. If you want to start the scan immediately in the default settings, you can click *Finish* button. We click the next button in this scenario.

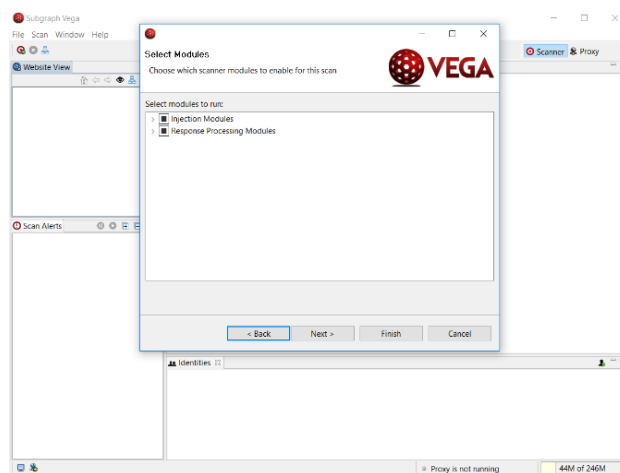


Figure 15. Selecting modules before scanning

After making the necessary adjustments click the next button. In this section, we click the finish button to start the scan and the scan is started.

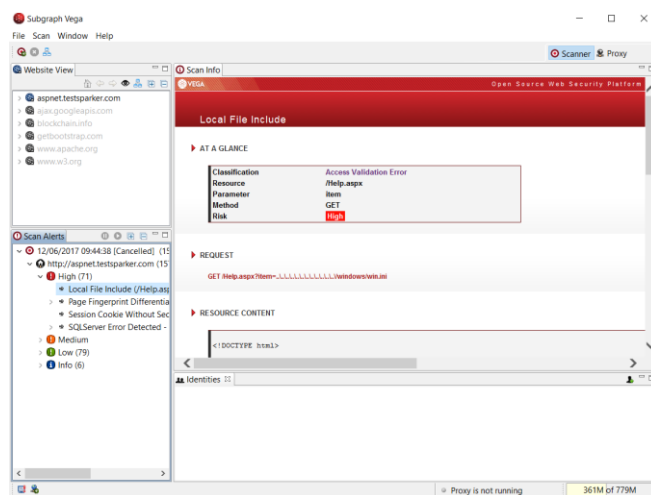


Figure 16. Entering the URL for scanning

Figure 16 shows the interface displayed after the scan is complete. The vulnerabilities found are listed on the left-hand side of the *scan alerts* window. When we click on any of the security vulnerabilities, detailed information and orientation comes to our main screen. As you can see in Figure 17, you can do the necessary research and close the vulnerabilities in your web app. Finally, you can review Vega's *Proxy* window to view the incoming requests and detailed proxy information. You can also initiate the http Proxy scan.

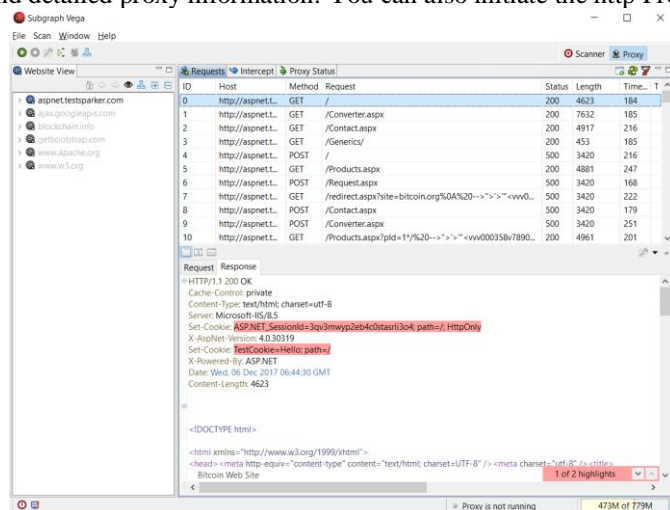


Figure 17. Checking detected vulnerabilities

Vega provides an advantage to its competitors with its open source and performance ratio. It is considered as a very good alternative for individual use in systems of lower severity. In this scenario, testing of the web app given in the scenario was completed in 45 minutes and found a total of 151 security vulnerabilities. In addition, the modules can be arranged or rewritten provides a great advantage to users.

#### D. OWASP Zed Attack Proxy (ZAP)

OWASP is a worldwide not-for-profit organization focused on improving the security of software. Its mission is to make software security visible, so that individuals and organizations are able to make informed decisions. OWASP is in a unique position to provide impartial, practical information about AppSec to individuals, corporations, universities, government agencies and other organizations worldwide. Operating as a community of like-minded professionals, OWASP issues software tools and knowledge-based documentation on application security. OWASP is also an online community that provides articles, methods, documentation, and tools freely in the field of web app security. ZAP is an easy-to-use, free and open source integrated penetration testing tool for detecting vulnerabilities in web apps. ZAP is actively supported by hundreds of international volunteers. ZAP is also known as OWASP's flagship project. It is an ideal tool for developers and functional testers who are new to penetration testing. ZAP also provides a set of tools that allow you to detect vulnerabilities manually in addition to automatic scanners. This makes it an excellent tool for experienced developers to use for manual security testing. ZAP can also produce reports in HTML and XML format [13].

#### Test Scenario;

The test was performed with OWASP ZAP 2.6.0 application on Kali Linux 2017.2 (x64).

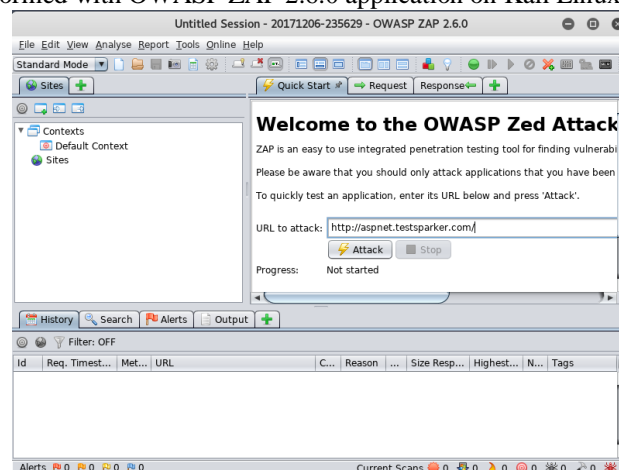


Figure 18. OWASP ZAP GUI and entering URL to attack

As shown in Figure 18, when we start the ZAP application, it provides us with a simple interface. Standard mode, attack mode, defense mode etc. modes are available in the program. We'll continue here with the standard mode. To start a new scan first, we write the URL of the target site to URL to attack in the main window. Then we click on the attack button to start the scan and the scan starts. Figure 19 shows the state of the related scan after completion.

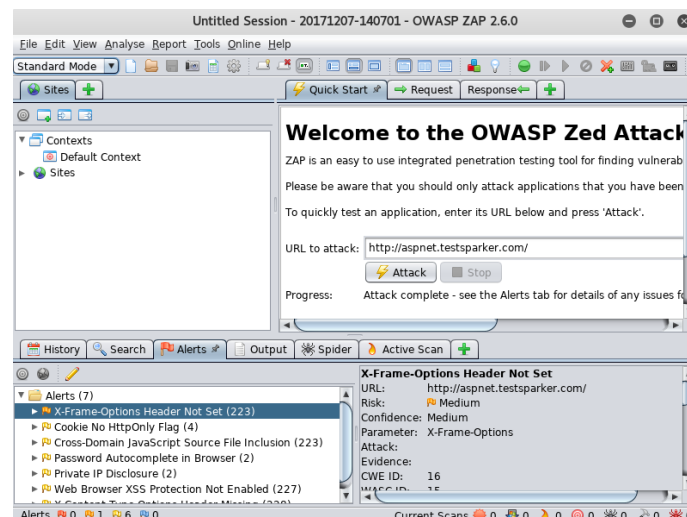


Figure 19. OWASP ZAP alerts after scanning

After the scan is completed, the results are sorted in the alert tab. When we click on any of the vulnerabilities found, it gives us detailed information about the description. If you want to print the report, from above you can click on the *Report* tab and get the output from the report as desired. On the History tab we can see the scans we've done before. In the *Search* section of our research, we can search for anything we are looking for. In the spider part we can reach the hidden content.

ZAP is advantageous because of open source based structure. But it does not detect all of the known security vulnerabilities in the system. This tool is considered to be more suitable for people new to penetration testing. In this scenario, the scan was completed in approximately 1.5 hours and detected a total of 7 vulnerabilities. One of the biggest advantages is the presence of a number of tools that allow you to manually find security vulnerabilities in addition to automatic scanners.

#### E. Wapiti

Wapiti is a free web security testing tool used to detect vulnerabilities in web apps. It performs a Black Box test, ie it does not analyze the source code of the application, but it scans the web pages of the web app to be tested and looks at scripts and forms that can inject data. Once it gets the list of URLs, forms and their inputs, Wapiti acts like a fuzzer, injecting payloads to see if a script is vulnerable.

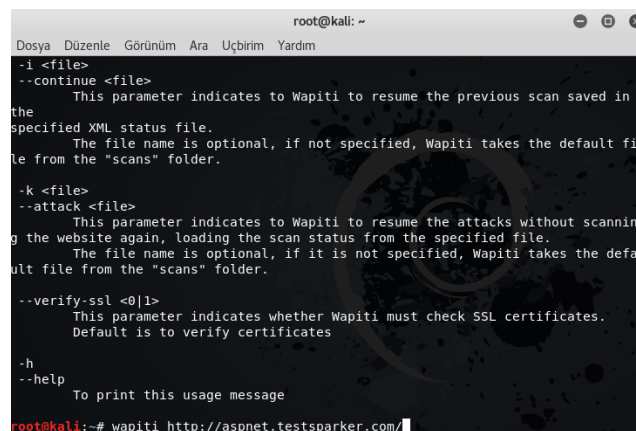
Wapiti can detect the following vulnerabilities:

- File disclosure (Local and remote include/require, fopen, readfile...)
- Database Injection (PHP/JSP/ASP SQL Injections and XPath Injections)
- XSS injection (reflected and permanent)
- Command Execution detection (eval(), system(), passtru()...)
- CRLF Injection (HTTP Response Splitting, session fixation...)
- XXE (XML External Entity) injection
- SSRF (Server Side Request Forgery)
- Use of know potentially dangerous files (thanks to the Nikto database)
- Weak .htaccess configurations that can be bypassed
- Presence of backup files giving sensitive information (source code disclosure)
- Shellshock (aka Bash bug)

Wapiti generates vulnerability reports in various formats (HTML, XML, JSON, TXT...). It can suspend and resume a scan or an attack (session mechanism using sqlite3 databases). It can give you colors in the terminal to highlight vulnerabilities. It provides different levels of verbosity. It is fast and easy way to activate/deactivate attack modules. Adding a payload can be as easy as adding a line to a text file. Wapiti supports GET and POST HTTP attack methods. Supports HTTP and HTTPS proxies. It provides full support for HTML5 [14].

Test Scenario:

The test was performed on Kali Linux 2017.2 (x64) with Wapiti 2.3.0 application.



```

root@kali: ~
Dosya Düzenle Görünüm Ara Uçbirim Yardım

-i <file>
--continue <file>
    This parameter indicates to Wapiti to resume the previous scan saved in
the specified XML status file.
    The file name is optional, if not specified, Wapiti takes the default fi
le from the "scans" folder.

-k <file>
--attack <file>
    This parameter indicates to Wapiti to resume the attacks without scannin
g the website again, loading the scan status from the specified file.
    The file name is optional, if it is not specified, Wapiti takes the defa
ult file from the "scans" folder.

--verify-ssl <0|1>
    This parameter indicates whether Wapiti must check SSL certificates.
    Default is to verify certificates

-h
--help
    To print this usage message

root@kali:~# wapiti http://aspnet.testsparker.com/
    
```

Figure 20. Vulnerability scanning with wapiti using Kali Linux operating system

When we start the Wapiti application, a terminal screen appears. Firstly, the parameters required to customize the scanning are displayed. To start the scan, type wapiti in the terminal, leave a space, write the URL of the

target site, and start the scan in the default settings. To shorten the scanning, you can leave spaces at the end of our scanning command and trigger only the modules we need, such as `-m, sql, blindsqli` (Figure 20).

```

Notice
=====
This scan has been saved in the file /root/scans/singhgurjot.wordpress.com.xml
You can use it to perform attacks without scanning again the web site with the "-k" parameter
[*] Loading modules :
    mod_crlf, mod_exec, mod_file, mod_sql, mod_xss, mod_backup, mod_htaccess
    , mod_blindsqli, mod_permanentxss, mod_nikto
[+] Launching module crlf
[+] Launching module exec
[+] Launching module file
[+] Launching module sql
[+] Launching module xss
[+] Launching module blindsqli
    
```

Figure 21. Vulnerability scanning with wapiti using Kali Linux operating system

As shown in Figure 21, when the scan is complete, detailed information about the scan has been reported to the file titled “index.html” into the path `/root/.wapiti/generated_report` in the HTML format. Information about the vulnerabilities found can be accessed from this file (Figure 21).

Due to the terminal-based operation of the application, it becomes difficult to use, especially in terms of ease of use, which is its competitors with a GUI. The ability to color the posts on the terminal increases the readability. Wapiti completed the scan in a very long time of approximately 13 hours and found a total of 129 vulnerabilities. To ensure that the scan does not take so long, only those you need can be run from the modules using special parameters for scanning. It is a nice feature to generate reports in HTML and XML formats. Scans made with Wapiti can be transferred to Metasploit and reports that you generate in XML format can be added to your Metasploit database. The fact that the application only performs Black Box testing, which does not allow us to analyze the source code of the application causes it to fall behind its competitors.

#### F. IronWASP

Iron Web Application Advanced Security Testing Platform (IronWASP) is a free and open source web security testing tool used to find vulnerabilities in web apps. It can identify more than 25 web vulnerabilities. It is a GUI based tool written in Python and Ruby. It can detect false positives and false negatives. IronWASP produces reports in HTML and RTF format. It can be expanded through plug-ins or modules written in Python, Ruby, C # or VB.NET [15].

#### Test Scenario:

The test was performed with IronWASP 2015 on Windows 10 (x64).

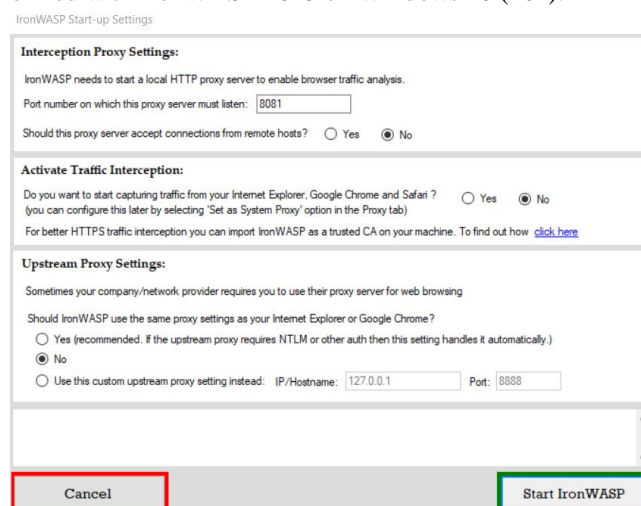


Figure 22. IronWASP graphical user interface

When we start the IronWASP application, the initial settings are coming. After making the necessary settings then click the *Start IronWASP* button.

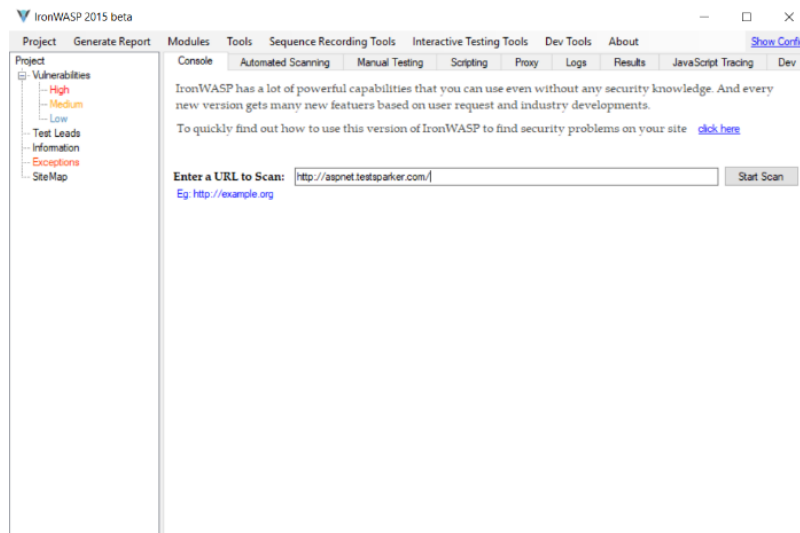


Figure 23. Starting to scan with IronWASP

When the application is opened, we come across a simple and simple interface. First, we write the URL of the destination site to *URL to Scan* in the main window to start a new scan. Then we click the *Start Scan* button to start the scan and the window opens.

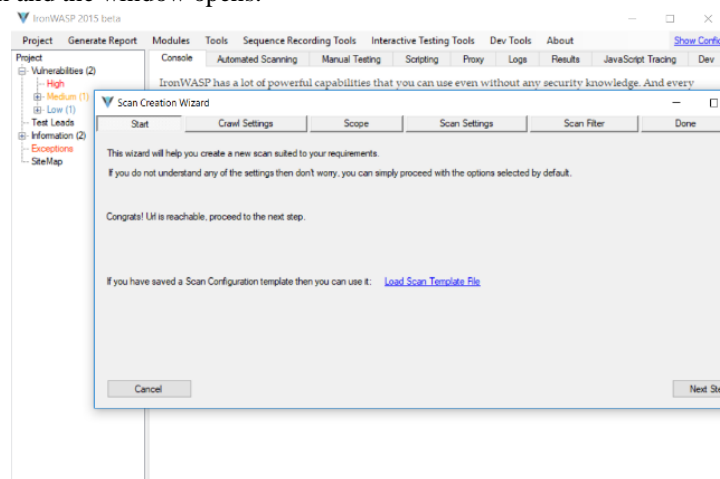


Figure 24. General settings with IronWASP scan wizard

In order to customize the scan we will make, the configuration settings are displayed on our screen. To make the necessary adjustments, click *Next Step* button. After making the necessary adjustments, click the *Start Scan* button and the scan starts.

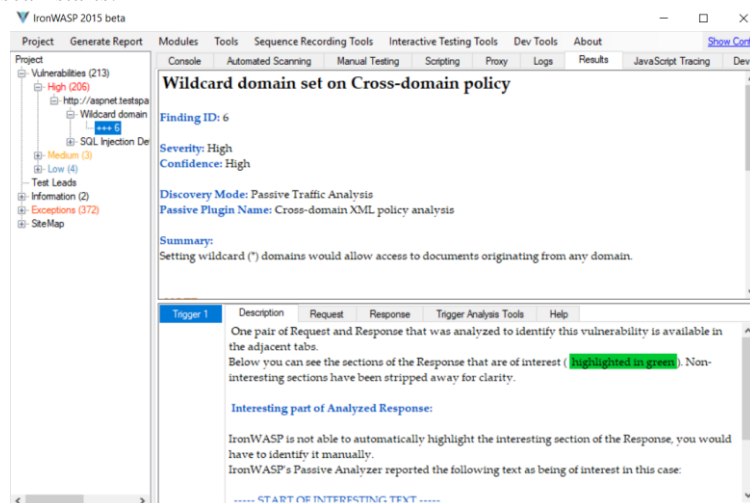


Figure 25. Detecting and reporting the vulnerabilities with IronWASP

The security vulnerabilities found were sorted by severity within the *Vulnerabilities* tab under the *Project* tab in the left-hand window when the scan was completed. When we click on any of the security vulnerabilities, detailed information and orientation comes to our main screen. You can make the necessary research in this section and close the security vulnerabilities in your web app. You can also get a detailed report of the scanning result when we click on the top of *Generate Report*.

IronWASP is an open source tool and is advantageous with its performance rate. The GUI-based IronWASP tool can generate reports in HTML and RTF format. IronWASP is easy to use, this making it a convenient tool for beginner pentesters. It's a good alternative for professionals with rich customization options. It is also suitable for both beginner and professional testers as well as individual use. In this test scenario, IronWASP completed the scanning in a very long time of 7 hours and detected a total of 213 vulnerabilities. The biggest disadvantages is the long scanning time. In addition, because of the modular structure of this tool, the modules can be arranged or rewritten to provide a great advantage to users.

## VI.EVALUATION BENCHMARKING AND CONCLUSION

In parallel with the widespread use of web apps, the need for web app tests is increasing. In addition to a functional specification, a test plan and test cases are required to check whether a vulnerability scanner provides a feature. A test plan details how a tool is tested, how the test results are interpreted, and how you summarize or report the tests. Currently, the vulnerability scanner tools produce reports in various formats. A common reporting format will make it easier to automate the comparison of different tools. We measure the suitability of a tool according to various test conditions. It is important to know how an attacker uses vulnerabilities when we select these test cases.

In a normal web app, a user sends a request to the web app and receives a response. But an attacker sends an unsuspected request to an application hoping to exploit an existing vulnerability. The purpose of an attacker is to violate the application's security policy. An attacker recognizes the existence of the vulnerability by examining the response of the application or noticing changes in the behavior of the application. The web app scanner works by simulating the attacker's action.

Web apps with security vulnerabilities are required to test web app scanners. There must be at least one test application for each class of vulnerability. Small test cases with a single vulnerability can be used to precisely test the ability of tools to detect specific vulnerabilities. It is also important to test the ability of tools to detect vulnerability in web apps created using different web technologies.

A basic test package can include applications that have vulnerabilities. For example, if an application does not verify any entries, there are many ways to exploit the vulnerability, and most tools can find. However, in order to thoroughly test a scanner, we need programs with sensitive vulnerabilities.

Different SQL injection types are another examples. An attacker usually sends a request that causes the application to create an SQL query that could cause unexpected behavior. Then, the attacker examines the error message that is returned to the web client. A typical mitigation approach is to prevent the application from displaying any database error messages.

In this study, various web app vulnerability scanners and web security testing tools have been examined and evaluated in detail. Suggestions are provided for which test tool is best suited for whom. The test tools provide information on how to perform various sample scenarios. In addition the advantages and disadvantages of the test tools are also indicated. A comparison of the data obtained from the applied scenarios and test tools is summarized in Table 1.

Table I. A Comparison of web vulnerability scanners

Name	Scanning Time	Vulnerabilities	Report	Module Based Structure	Manually Test Tools	GUI	Operating Systems	Fee	Using Area
Netsparker	2 hours	47	Yes	No	Yes	Yes	Linux, Windows, Mac OS	5000 \$	Institutional
Acunetix	2.5 hours	163	Yes	No	Yes	Yes	Linux, Windows, Mac OS	4500 \$	Institutional
Vega	45 minutes	151	Yes	Yes	No	Yes	Linux, Windows	Free	Individual
OWASP ZAP	1.5 hours	7	Yes	No	Yes	Yes	Linux, Windows, Mac OS	Free	Individual
Wapiti	13 hours	129	Yes	Yes	Yes	No	Linux, Windows	Free	Individual
IronWASP	7 hours	213	Yes	Yes	Yes	Yes	Linux, Windows, Mac OS	Free	Individual

## REFERENCES

- [1] Fortune Turkey, [Online]. Available: <http://www.fortuneturkey.com/fotohaber/dunyanin-en-degerli-markalari-2017-42674>.
- [2] Jeremiah Grossman, The Five Myths of Web Application Security, WhiteHat Security, Inc, 2005.
- [3] Prescatore, John, Gartner, Computerworld, 25 Şubat 2005, [Online]. Available: <http://www.computerworld.com/printthis/2005/0,4814,99981,00.html>.
- [4] G. McGraw, Yazılım Güvenliği: Bina Güvenliği, Addison-Wesley Software Security Series, 2006.
- [5] Web Uygulaması Güvenlik Konsorsiyumu Sözlüğü, [Online]. Available: <http://www.webappsec.org/projects/glossary/>.
- [6] Jody Melbourne ve David Jorm, Penetration Test for Web Applications, SecurityFocus, 2003.
- [7] Web Application Security Consortium, "Threat Classification", [Online]. Available: <http://www.webappsec.org/projects/threat/>.
- [8] OWASP, "OWASP Top 10 Most Critical Web Application Security Risks", [Online]. Available: [http://www.owasp.org/index.php/OWASP\\_Top\\_Ten\\_Project](http://www.owasp.org/index.php/OWASP_Top_Ten_Project).
- [9] Common Weakness Enumeration: CWE, MITRE, <http://cve.mitre.org/cwe/>.
- [10] Netsparker, Web Application Security Scanner, <https://www.netsparker.com/>.
- [11] Acunetix, Web Site Security, <https://www.acunetix.com/>.
- [12] Vega, Vulnerability Scanner, <https://subgraph.com/vega/index.fr.html>.
- [13] OWASP ZAP, Zed Attack Proxy, [https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project).
- [14] Wapiti, Web Vulnerability Scanner, <http://wapiti.sourceforge.net/>.
- [15] IronWASP, Application Security Scanner, <https://www.utest.com/tools/ironwasp>.