

**Application Security: Vulnerability Assessment – Mutillidae / OWASP™-Zap / Nikto**

**James Finglas**

**B00094138**

Department of Informatics

School of Informatics and Engineering

Technological University Dublin

Submitted to Technological University Dublin in Partial Fulfilment of the requirements for the degree of:

Bachelor of Science (Honors) in Computing

Course: Application Security

Lecturer: Stephen O'Shaughnessy

Submission Date: 1/12/2019

Word count (Introduction to Conclusions): 7667

# Project Submission/Declaration Form

Submitted by: James Finglas

Course Coordinator: Stephen O'Shaughnessy

Module Name: Application Security

Course Code: H4013

Lecturer: Stephen O'Shaughnessy

Submission Date: 1/12/2019

I declare that the work contained in this report which I now submit on the program of study leading to the award of Degree of Honors B.Sc. in Computer Science in Technological University for Dublin is my own work and has not been taken from the work of others. Any sources which have been referenced or analyzed have been cited using the Harvard/IEEE standard within the body of this report. It is entirely my own work except where otherwise stated and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilments of the requirements stated above.

I have read and understood the Technology University Dublin’s policy regarding plagiarism.

X

James Finglas

# Acknowledgements

Since this is the last major report of this term, I would like to take a moment to thank, and acknowledge the work of my lecturer and course coordinator this year, Stephen O'Shaughnessy. His advice to really set aside time to practice our abilities to find, diagnose, evaluate vulnerabilities and exploit them.

The module was a slow burner. Enjoyable with almost no pressure being applied with the spacing of assignments. The assignments were tough, yet deeply enjoyable and what I have learned are some incredibly useful skills that I will absolutely take with me, develop further and deploy in my professional life. The python classes were incredible and in just a few short weeks I rediscovered an interest in programming I had almost lost touch with. The module has been a blast, one of the best of my entire time thus far in University; and I have rarely felt as confident and prepared approaching assignments as I have in this module. To teach a student is a given, but to re-invigorate a student’s love of a subject in which his interest had flagged, while also teaching him incredibly valuable skills that will defiantly be utilized is a special gift.

Thank you.

# Abstract

For our final major assignment in Application Security, we were tasked with carrying out a simulated vulnerability assessment. While it is not possible to conduct such an assessment truly as if it were real world, we must try to simulate it where and when we can as though it were real world.

We were tasked with selecting a vulnerable web application and a minimum of 1 vulnerability scanner. We then had to screen for as many vulnerabilities as possible and filter down the technical reports to a much more readable format that a non-technically skilled client would be able to read and evaluate for the purposes of carrying out assets and costs risk assessments to the best of his/her ability.  
  
Mitigations were to be offered for each assessment which I have documented, alongside selected demonstrations of exploits. For my vulnerable app I chose ‘Mutillidae’ running on the ‘Metasploitable 2.0’ VM platform, with security Onion acting as a passthrough should further network diagnostics, analytics or manipulations be required. For my Scanner I elected to run two scanners, Nikto and OWASP™-Zap. My selection choices have been documented and explained within the methodology section of this report.  
  
This Report documents an introduction to the assignment, followed by a small summary broken down into a research goals, the scope of the assessment and the methodology employed sections followed by the primary research outcomes consisting of my Nikto and OWASP™-ZAP scan results.

Having far too many exploits to demonstrate them all, I elected to aim for approximately 5% of my total exploits; spreading them over as many types of vulnerabilities as possible. The steps taken and the results are documented in the primary research section of this report.

Lastly, we were tasked with drawing some conclusions from our report and giving and final recommendations we feel would be of benefit to our client during the implementation of any mitigations and remediations we have recommended. I present my report assessment and evaluation.

Contents

[Project Submission/Declaration Form 2](#_Toc26013336)

[Acknowledgements 3](#_Toc26013337)

[Abstract 4](#_Toc26013338)

[Table of figures: 6](#_Toc26013339)

[Table of Tables: 7](#_Toc26013340)

[1.0 Introduction 9](#_Toc26013341)

[2.0 Summary 10](#_Toc26013342)

[2.1 Research Goals 10](#_Toc26013343)

[2.2 Scope 10](#_Toc26013344)

[2.3 Methodology 11](#_Toc26013345)

[3.0 Primary Research Outcomes 16](#_Toc26013346)

[3.1 Nikto Scan Results 16](#_Toc26013347)

[3.2 OWASP™-Zap Scan Results 30](#_Toc26013348)

[5.0 Conclusions 50](#_Toc26013349)

[6.0 Glossary 51](#_Toc26013350)

[7.0 References 53](#_Toc26013351)

# Table of figures:

[Figure 1 My attack machine details. 10](#_Toc25969503)

[Figure 2 My selected web app scanning tool, OWASP™-Zap v2.6.0. 10](#_Toc25969504)

[Figure 3 My live scan, having just been initiated in 'ATTACK' mode. 12](#_Toc25969505)

[Figure 4 Here we see my 5 scans which concurrently in a single session. 13](#_Toc25969506)

[Figure 5 Here we can see some sample output from my final Nikto scan. 13](#_Toc25969507)

[Figure 6 Some sample output from my initial OWASP™-Zap report. 14](#_Toc25969508)

[Figure 7 Here we see access to a file exposed via robots.txt 18](#_Toc25969509)

[Figure 8 This file contains the entire password list. 19](#_Toc25969510)

[Figure 9 An alternate database password revealed for another OS. 19](#_Toc25969511)

[Figure 10 Directory structure, and cross site scripting framework open to exposure and manipulation. 19](#_Toc25969512)

[Figure 11 JavaScript scripts open to exposure and manipulation. 20](#_Toc25969513)

[Figure 12 Entire website tree structure exposed. 20](#_Toc25969514)

[Figure 13 Another page exposed, although this page is purposefully exposed as part of the legitimate working of this purposefully vulnerable app. 21](#_Toc25969515)

[Figure 14 First i entered a single quote in an attempt to illicit an SQL database error. 34](#_Toc25969516)

[Figure 15 And here we have our desired SQL error revealed the required search syntax. 34](#_Toc25969517)

[Figure 16 Here we attempt to bypass the authentication by injecting a Boolean true statement into the hard-coded SQL search string. 35](#_Toc25969518)

[Figure 17 And we succeeded and are logged in as Admin. 35](#_Toc25969519)

[Figure 18 Here we see me entering my simple XSS script into the user input text box. 36](#_Toc25969520)

[Figure 19 My script is immediately executed by the server. 37](#_Toc25969521)

[Figure 20 Finally, the now persistent XSS attack is launched. 37](#_Toc25969522)

[Figure 21 Here have a successful shell creation. 42](#_Toc25969523)

[Figure 22 Root access and full system compromise. 43](#_Toc25969524)

[Figure 23 Nmap results from our scan of our informational leaked private IP addresses. 48](#_Toc25969525)

# Table of Tables:

[Table 1 Nikto Vulnerability 1 15](#_Toc25969960)

[Table 2 Nikto Vulnerability 2. 15](#_Toc25969961)

[Table 3 Nikto Vulnerability 3. 16](#_Toc25969962)

[Table 4 Nikto Vulnerability 4. 16](#_Toc25969963)

[Table 5 Nikto Vulnerability 5. 16](#_Toc25969964)

[Table 6 Nikto Vulnerability 6. 17](#_Toc25969965)

[Table 7 Nikto Vulnerability 7. 17](#_Toc25969966)

[Table 8 Nikto Vulnerability 8. 17](#_Toc25969967)

[Table 9 Nikto Vulnerability 9. 17](#_Toc25969968)

[Table 10 Nikto Vulnerability 10. 18](#_Toc25969969)

[Table 11 Nikto Vulnerability 11. 18](#_Toc25969970)

[Table 12 Nikto vulnerability 12. 21](#_Toc25969971)

[Table 13 Nikto Vulnerability 13. 22](#_Toc25969972)

[Table 14 Nikto Vulnerability 14. 22](#_Toc25969973)

[Table 15 Nikto Vulnerability 15. 22](#_Toc25969974)

[Table 16 Nikto Vulnerability 16. 22](#_Toc25969975)

[Table 17 Nikto Vulnerability 17. 23](#_Toc25969976)

[Table 18 Nikto Vulnerability 18. 23](#_Toc25969977)

[Table 19 Nikto Vulnerability 19. 23](#_Toc25969978)

[Table 20 Nikto Vulnerability 20. 24](#_Toc25969979)

[Table 21 Nikto Vulnerability 21. 24](#_Toc25969980)

[Table 22 Nikto Vulnerability 22. 24](#_Toc25969981)

[Table 23 Nikto vulnerability 23. 25](#_Toc25969982)

[Table 24 Nikto vulnerability 24. 25](#_Toc25969983)

[Table 25 Nikto vulnerability 25. 25](#_Toc25969984)

[Table 26Nikto Vulnerability 26. 26](#_Toc25969985)

[Table 27 Nikto vulnerability 27. 26](#_Toc25969986)

[Table 28 Nikto Vulnerability 28. 26](#_Toc25969987)

[Table 29 Nikto Vulnerability 29. 26](#_Toc25969988)

[Table 30 Nikto vulnerability 30. 27](#_Toc25969989)

[Table 31 Nikto vulnerability 31 27](#_Toc25969990)

[Table 32 Nikto vulnerability 32 27](#_Toc25969991)

[Table 33 OWASP™-Zap vulnerability 1. 29](#_Toc25969992)

[Table 34 OWASP™-Zap vulnerability 2. 30](#_Toc25969993)

[Table 35 OWASP™-Zap vulnerability 2 (continued). 31](#_Toc25969994)

[Table 36 OWASP™-Zap vulnerability 3. 32](#_Toc25969995)

[Table 37 OWASP™-Zap vulnerability 4. 33](#_Toc25969996)

[Table 38 OWASP™-Zap vulnerability 5. 36](#_Toc25969997)

[Table 39 OWASP™-Zap vulnerability 6. 38](#_Toc25969998)

[Table 40 OWASP™-Zap vulnerability 7. 38](#_Toc25969999)

[Table 41 OWASP™-Zap vulnerability 8. 39](#_Toc25970000)

[Table 42 OWASP™-Zap vulnerability 9. 40](#_Toc25970001)

[Table 43 OWASP™-Zap vulnerability 10. 40](#_Toc25970002)

[Table 44 OWASP™-Zap vulnerability 11. 41](#_Toc25970003)

[Table 45 OWASP™-Zap vulnerability 12. 44](#_Toc25970004)

[Table 46 OWASP™-Zap vulnerability 13. 45](#_Toc25970005)

[Table 47 OWASP™-Zap vulnerability 14. 46](#_Toc25970006)

[Table 48 OWASP™-Zap vulnerability 15. 47](#_Toc25970007)

[Table 49 OWASP™-Zap vulnerability 16. 47](#_Toc25970008)

# Introduction

In the modern world of Information technology, security and interconnected networks, application security is of paramount concern to any person or agency that produces an application that may be accessed by the public, as discussed by IBM in a 2018 white paper titled ‘Five Steps to achieve Risk-Based Application Security Management’ [1].

The is where vulnerability testing comes in. Much like a penetration test, a web-based vulnerability test is designed to highlight vulnerabilities across the attack-surface of the website or web application. Vulnerability testing is crucial and should be considered part of the modern software development life cycle and carried out through the life of the application periodically. Under the CMMI-DEV framework developed by Carangie Mellon University in 1987, this vulnerability assessment would fall into the category of ‘Security Verification and Validations (SVV)(Engineering)’, as referenced by Barbara Filkens in her Sans Institute white paper ‘Security by Design: The Role of Vulnerability Scanning in Web App Security’ [2].

A vulnerability scanner is used to attempt to enumerate webpages, access pages that should be restricted, check for vulnerabilities in the software service running the web application and attempt to highlight ways that an attack agent might be able to exploit the web application for malicious purposes, such as pivoting to the OS of the machine running the web application.

This is what we have been tasked with. First, we must source a vulnerable web application, preferably one recognized by OWASP. Next, we must select an appropriate web scanner and use the scanner to assess the level of vulnerability our chosen web application. This demonstrates our ability to carry out one of the most basic functions of a information security red team penetration tester or blue team developer who may be attempting to fix vulnerabilities.

We must document these vulnerabilities, as well as a descriptive breakdown of the vulnerability. We may attempt to exploit these vulnerabilities and any successes we may have in exploitation should also be documented. However, should there be replication of vulnerabilities throughout the web application, each vulnerability should only be tested once. An impact assessment will also be carried out and documented to weight up the potential affect, both in terms damage or exposure of data, and estimated level of cost to the client.  
  
This is a crucial part of a vulnerability assessment, as it is not important to only fix a vulnerability, but to highlight the risks and costs of such vulnerabilities so that can be avoided going forward in the future, This not only reduces the risks of the current vulnerability but also reduces the likely hood of vulnerability replication in any future additions to the web application as discussed in the ISACA white paper, 2017, titled ‘Vulnerability Assessment’ [3]. Once the blue team developers are aware of a vulnerability and the recommended mitigation, they should implement it, and the vulnerability should not be repeated because code implementation frameworks have been designed to prohibit its replication. This framework is behind the scope of the vulnerability assessment but is born of the assessment results.  
  
Having completed our assessment and documentation of its results, we must then draw conclusions from the process.

# Summary

## 2.1 Research Goals

The goal of this assignment is to demonstrate our ability to deploy scanning tools and discover vulnerabilities within web applications. This is one of the fundamental roles of a Security operations center analyst or penetration tester / researcher / threat hunter. A blue team developer might also be required to carry out these tests during the software design lifecycle process and during regular maintenance to test a recently implemented mitigation of a previously discovered vulnerability.

We also where afforded a chance to display our ability to carry out certain attacks by deploying tools such as ‘BurpSuite., ‘Wireshark’, ‘Metasploitable’, and client-side injection attacks against our chosen web applications. For the purposes of expediency, I shall limit myself to 5 demonstrations of exploits as my technical document illustrates the exploits.  
  
Lastly, we had the opportunity to display our ability to present the gathered information in the format of a professional, clear and coherent readable manner.

## 2.2 Scope

The vulnerable web app shall be considered in scope for this assessment. The host, in so far as any vulnerabilities that may affect the running of the app directly shall also be considered in Scope. For example, should the program language used to script the app, or any language implemented by the app or installed that could affect app; it too shall be deemed in scope

The host operating system shall be considered out of scope. As this entire platform is a vulnerable platform and we would find many vulnerabilities within it, documenting all of these would make the document prohibitively large. It is acknowledged that in a real-world scenario, ANY vulnerability that might result in the host system being compromised constitutes a threat to the application, but we shall treat the assignment as though we have been provided a scope that prohibits any such deeper diving into the host itself.  
  
However, services running on the host shall be deemed in scope if their being compromised could affect the web app.

Lastly, the client’s network, be it up-stream or down-stream of the web app host, for the purposes of documenting any potential exposures, is considered in scope for limited scanning. Not active penetration of any host beyond the web app host machine is in scope and is strictly prohibited.

## 2.3 Methodology

I feel it important to stress that in a real-world scenario I would never restrict myself to just 1 or even 2 scanners. I would deploy many such as ‘Open-Vas’, ‘Nessus’, ‘Nikto’, ‘Wmap’, ‘BurpSuite’, ‘Edge Scan’, ‘Net Sparker’, ‘OWASP™-Zap’, etc.; cross-referencing the results and collate all the reports into one report for the client’s digestion.

However, to preserve the sanity of my lecturer, and for the purposes of this assignment; I will reserve myself to two and I have selected a vulnerable app which I feel has an appropriate amount of vulnerabilities without being too excessive.

I elected to use the OWASP vulnerable web app called ‘Mutillidae’ version 2.1.19 running on the Metasploitable2.0 virtual machine [4]. For my Attack/Scanning machine I chose Kali.

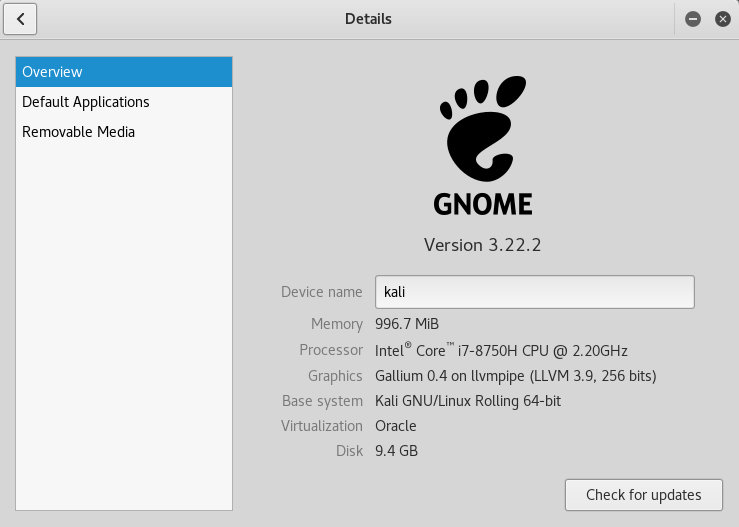


Figure 1 My attack machine details.

For scanning I selected OWASP-Zap™ version 2.6.0 as my scanning tool as it goes hand in hand with the OWASP™ vulnerable web app; and I combined this with Nikto as a secondary scanner for thoroughness [5][6]. Nikto Is a useful scanner which highlights CVE vulnerabilities as well as fingerprinting the host and collects versioning, OS and other fingerprint related information which OWASP™-Zap does not focus on.

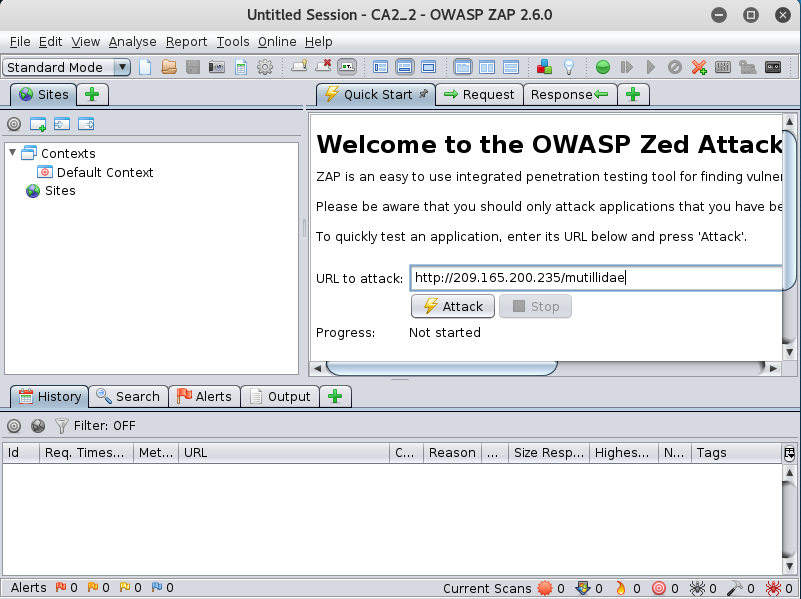


Figure 2 My selected web app scanning tool, OWASP™-Zap v2.6.0.

As a secondary cross reference tool, I have used the network analysis and diagnostic Security Onion OS as a pass through. This gives me access to ELSA, SQUIL, BRO, SQUERT, Wireshark and other powerful diagnostic traffic tools. This may not be used but the optional data mining tools to view attacks in progress is useful to have available.

OWASP™-Zap Offers 4 scanning options:

* Safe Mode: Safe mode is essentially a much less noisy scanning mode that does not allow the user to carry out any attacks or tests that might be considered ‘Dangerous’.
* Protected Mode: This Mode allows the user to simulate attacks, but only if a scanned page is deemed in scope.

* Standard Mode: This is the most common mode, used during software development life cycles and often on live applications in order to scan and find vulnerabilities but minimize the risk of taking the site offline and causing downtime.
* Attack Mode: This is an aggressive noisy scan, which attempts to attack any node/URL immediately upon discovery, and should any changes be made during the scan, the node/URL will be rescanned.

Even though I ran my web app on a sandboxed VM and knew there would be no activity on the web application apart from my active scan, I nonetheless ran my scan in ‘ATTACK’ mode. This would be my go too choice, if I where deploying the tool in a real-world scenario. This would be to test how vulnerable a web app is and how robust it is under the load of a heavy screening/attack process. Generally, I would couple this with a simulated progressively increasing dos attack to attempt to ascertain the breaking point of the web apps traffic bandwidth to enable proper calibration of load-balancing. This is beyond the scope of this course; so, I reserved my process to scanning.

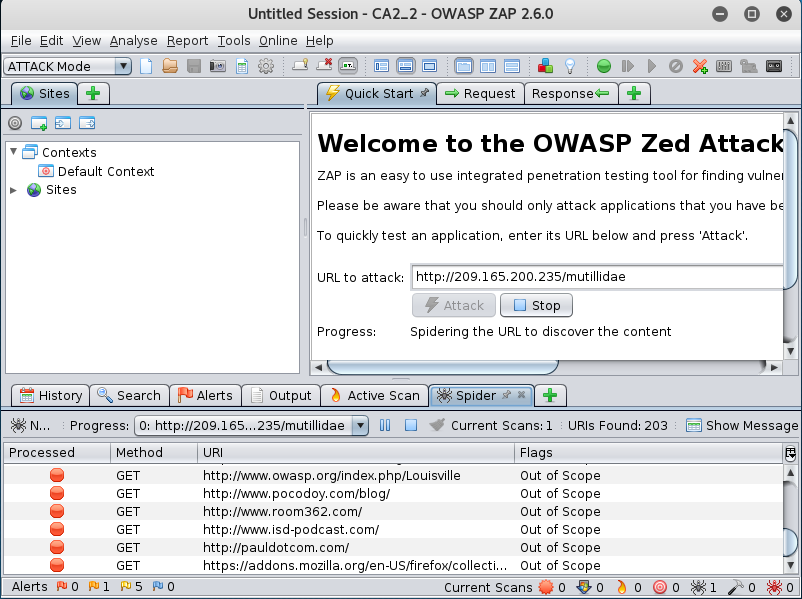


Figure 3 My live scan, having just been initiated in 'ATTACK' mode.

Above, you can see my first live attack having been initiated is in the spidering stage, screening for nodes/URL’s. Already we can see 5 minor alerts triggers and 1 medium alert triggered. Furthermore, we can see 207 URL’s have already been detected.

I ran 5 active ‘ATTACK’ mode scans with OWASP™-ZAP and 5 scans with Nikto which I pipped to a text file. My first pass with OWASP™-Zap revealed 7 high level alerts, 3 medium-level alerts and 5 low-level alerts. However, my second pass increased the medium alerts to 4 with the low-level alerts remaining consistent. These numbers all remained consistent over the next 3 scans. This indicates my first scan had a high-level false positive which was adjust in the second scan, highlighting the value of multiple cross-referencing scans. My Nikto scans proved very useful and remaining consistent with each scan. The total OWASP™-Zap results after 5 scans over 1 persistent session was 7 High 4 medium and 5 low risk alerts with 0 informational across all 5, however, the total from nikto was over 30 results.   
  
This again highlights the importance of scans across the entire attack surface encompassing multiple scanners. Had I restricted myself to 1, I would have missed a multitude of vulnerabilities.

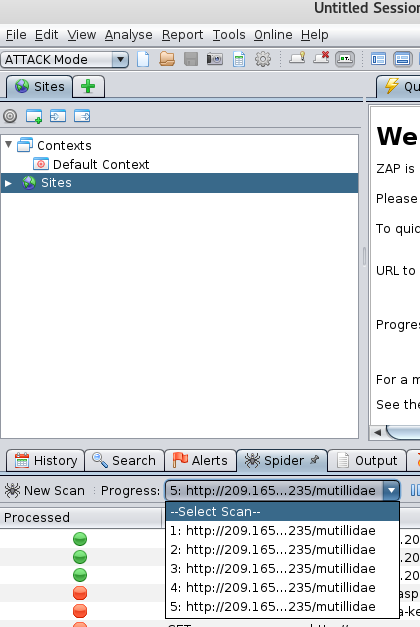


Figure 4 Here we see my 5 scans which concurrently in a single session.

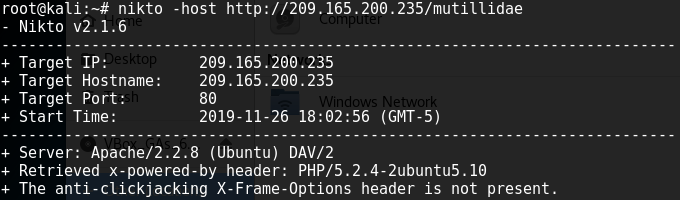


Figure 5 Here we can see some sample output from my final Nikto scan.

Once my scans completed, I transferred the analytical data to a html document, to allow to manually document my results in my report; a sample of which can be seen below.

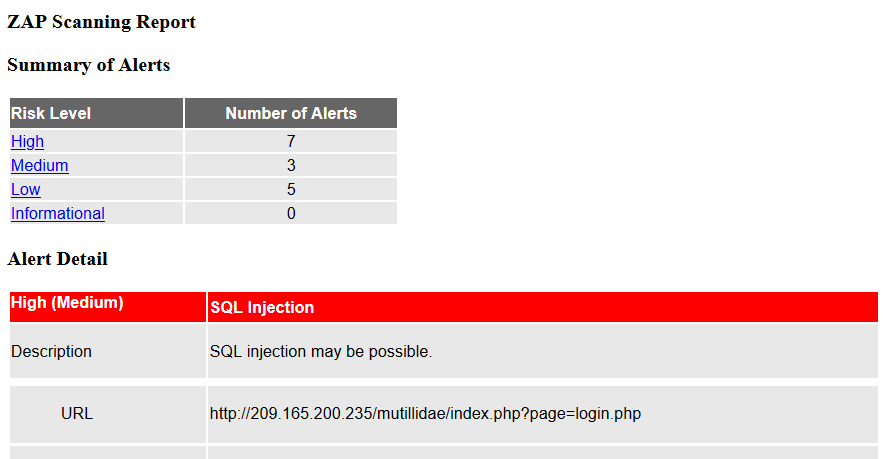


Figure 6 Some sample output from my initial OWASP™-Zap report.

# Primary Research Outcomes

## 3.1 Nikto Scan Results

Nikto outputs its results via CLI output in raw text format. The following are the results of my 5 scans pipped into a text document, however, I have compiled the data into a more readable format and included pertinent data related to vulnerabilities or CVE’s and possible Mitigations. Some technical data has been removed, however my technical reports will be provided along with the document.

Nikto Version: 2.1.6

Target IP / Hostname: 209.165.200.235

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| Target Port: 80 | This tells us no SSL certificate is being used. All data has the potential to be intercepted and read in plain text by a man in the middle threat actor. Password could be intercepted, as well as links to admin pages and other sensitive information. This could easily result in full system compromise. | This should be rectified by adding an SSL certificate and routing the web application through port 443 adding basic encryption to all data in transit. |
| Threat-Level: High | | |

Table 1 Nikto Vulnerability 1

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| Server: Apache/2.2.8 (Ubuntu) DAV/2. Apache/2.2.8 appears to be outdated (current is at least Apache/2.4.12). Apache 2.0.65 (final release). | This is an example of the risk related to hardware and software information leakage. Here we have the server version returned as 2.2.8, and we search the Apache projects vulnerability section and find 50 CVEs who’s affected versions denote 2.2.8. | All attempts to prevent information leakage should be made. However, this one may be unavoidable; therefore, server software must be kept uptodate as all 50 of the CVE’s mentioned have been patched over time. New CVE’s will continue to emerge and be patched, hence the need for constant upgrading. |
| Threat-Level: High | | |

Table 2 Nikto Vulnerability 2.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| Service detected: PHP/5.2.4-2ubuntu5.10 | Information leakage revealing the name of the Programming language with version and the OS with version. % exploits are revealed for the program language version, with 6 being revealed for the OS. | This can be mitigated with regular update maintenance across the platform, from host, to program language, server version and everything in between. |
| Threat-Level: High | | |

Table 3 Nikto Vulnerability 3.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| The anti-clickjacking X-Frame-Options header is not present. | This potentially allows a threat actor to inject click jacking frames which could redirect users to illicit websites which could capture credentials or user data. This can also be used to inject JavaScript executables onto user’s machines [7]. | This can be mitigated by implementing the following X-Frame options:  X-Frame-Options: SAMEORIGIN (This allows only the source website to create and embed frames)  X-Frame-Options: DENY (This denies all embedded frames, but is restrictive and can disable some desired functionality)  X-FrameOptions: ALLOW-FROM [IP / Domain Name] (This allows explicit allow rules for embedding of frames)  Example Implementation: Add ‘header always set x-frame-options “SAMEORIGIN” always;’ to httpd.conf of Apache2.0 config file. |
| Threat-Level: Medium | | |

Table 4 Nikto Vulnerability 4.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS | This tells any MITM attack agent that XSS protection is not enabled and the web app may be vulnerable to cross site scripting attacks [8]. Risk vary from client side injection to full system compromise. | This is mitigated by implementing the desired level of X-XSS-Protection Header such as X-XSS-Protection:0;, X-XSS-Protection:1;, X-XSS-Protection:1; mode=block  Example Implementation: Add ‘header always set x-xss-protection “1; mode=block” always;’ to httpd.conf of Apache2.0 config file. |
| Threat-Level: High | | |

Table 5 Nikto Vulnerability 5.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| Uncommon header 'logged-in-user' found, with contents: | Possible False-Positive. The risk here is cost effective. False positives increase the cost of both analysis during the SDLC, and future analytics. | Requires further investigation by blue team developers or tier 2 analyst support to determine if false positive. |
| Threat-Level: Low | | |

Table 6 Nikto Vulnerability 6.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| The X-Content-Type-Options header is not set. | This could allow the user agent to render the content of the site in a different fashion to the MIME type.  Essentially this amounts to a sniffing attack which data mines the server. File types can be mined allowing attack agents to initiate XSS attacks against assets [9]. | It should be noted that this mitigation is not currently honored by all vendors. Google Chrome & Some version of Internet Explorer currently honor this mitigation.  Example mitigation: Add ‘add\_header X-Content-Type-Options "nosniff"’ to httpd.conf of Apache2.0 config file. |
| Threat-Level: Medium | | |

Table 7 Nikto Vulnerability 7.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| Cookie PHPSESSID created without the httponly flag | Lack of the http-only flag during cookie creation exposes the cookie to the potential of client-side scripting capturing the cookie which could lead to session hijacking attacks [10]. | Apply the HttpOnly flag to all create cookies.  Example mitigation:  Set-Cookie: <name>=<value>[; <Max-Age>=<age>]  [; expires=<date>][; domain=<domain\_name>]  [; path=<some\_path>][; secure][; HttpOnly] |
| Threat-Level: High | | |

Table 8 Nikto Vulnerability 8.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| No CGI Directories found (use '-C all' to force check all possible dirs) | No risk here. This is positive output informing the user agent that no CLI interfaces or compiled scripts such as Perl scripts have been found running within the web app. | n/a |
| Threat-Level: Low | | |

Table 9 Nikto Vulnerability 9.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| Server leaks inodes via ETags, header found with file /mutillidae/robots.txt, 92442, size: 160, mtime: Tue May 10 17:00:04 2011 | This is a false positive caused by the addition of a ‘-‘ within the etag contained on the Robots.txt page [11]. | While no mitigation is strictly needed here, it can be mitigated by remove the ‘-‘ from the etag associated on this page. |
| Threat-Level: Low | | |

Table 10 Nikto Vulnerability 10.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| "robots.txt" contains 6 entries which should be manually viewed. These pages include:   * ./passwords/ * ./config.inc * ./javascript/ * ./wasp-esapi-php/ * ./documentation/ | Robots.txt should not provide links to pages that contain unprotected data, downloads links etc.  The pages contain within the web apps robots.txt page lay the entire web app open to being decimated by attack agents. It’s important to remember that this is a purposefully vulnerable app, and this is unlikely to happen to this extent in reality, but it highlights the damage that can be done. \*see screenshots below to gauge the extent of the exposure in this case\* | Do not post obfuscated links to sensitive data which are not protected via passwords. The passwords.txt page is one of the first pages manually viewed by any attack agent. |
| Threat-Level: High | | |

Table 11 Nikto Vulnerability 11.

**Exploit Demonstration:**

As seen above, the robots .txt leaks critically sensitive information which lays the application bare for attacks to fully compromise the application in seconds leaking passwords, scripts, images, links, directory tree structures and all pertinent sensitive data pertaining to the application. Which I have documented via images below.



Figure 7 Here we see access to a file exposed via robots.txt

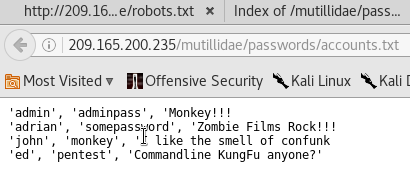


Figure 8 This file contains the entire password list.

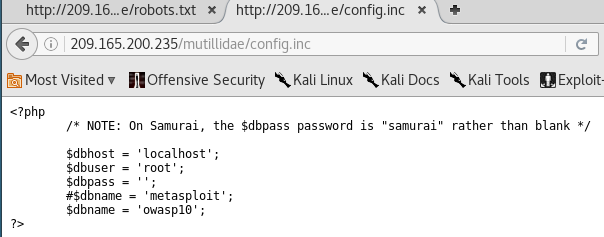


Figure 9 An alternate database password revealed for another OS.



Figure 10 Directory structure, and cross site scripting framework open to exposure and manipulation.



Figure 11 JavaScript scripts open to exposure and manipulation.



Figure 12 Entire website tree structure exposed.



Figure 13 Another page exposed, although this page is purposefully exposed as part of the legitimate working of this purposefully vulnerable app.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| Uncommon header 'tcn' found, with contents: list | This is a header which translates a portion of the URL into randomly generated txt. This is an attempt to prevent page enumeration. However, a highly skilled attack agent can use the tcn header, via ‘socat’ to gain access to the server, including SSH logins [12]. | Do not implement list option with the tcn header. |
| Threat-Level: Medium | | |

Table 12 Nikto vulnerability 12.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| Apache mod\_negotiation  is enabled with MultiViews,  which allows attackers to  easily brute force file names.  See http://www.wisec.it/sectou.  php?id=4698ebdc59d15. The following alternatives for 'index' were found: index.php | The mod negotiation function serves to select from a list, where a list of suitable pages is available. For example, a web app may have a backup index page, for development purposes which is not supposed to be displayed but can be used for testing functions. This page may contain sensitive information relating to server config and hardware as well similar software information [13]. | Disable the mutliview option in the Apache config options files and restart Apache2.0. |
| Threat-level: Medium | | |

Table 13 Nikto Vulnerability 13.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| Site appears to be vulnerable to the ‘shellshock vulnerability’ | This refers to CVE-2014-6271. This is a bash execution exploit that exposes DHCP clients, OpenSHH sshd, mod\_cgi, mod\_cgid and Apache2.0 http modules to exploitation. [14] Impact can be as high as total system compromise. | A patch was released for this exploit and found to be ineffective and a second patch was released, this is the patch that should be applied. As documented by CVE-2014-7169 [15]. |
| Threat-Level: Medium | **Applies to:**   * index.php * login.php | |

Table 14 Nikto Vulnerability 14.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| Allowed HTTP Methods: GET, HEAD, POST, OPTIONS, TRACE | While most of the http methods allowed are ok, the TRACE method can be used to initiate a cross site tracing attack [16]. | Remove the TRACE method from http config. |
| Threat-Level: Medium | | |

Table 15 Nikto Vulnerability 15.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| Web Server returns a valid response with junk HTTP methods, this may cause false positives. | This indicates the site is using a URL routing function such as angular, ng-route or $.route/$.Observe. This could potentially be used to leak routing information or even suffer injection to redirect traffic [17]. | All user inputs should be sanitized. The exact sanitization method will be up to the blue team developer to decide what is best such as html snippets or prepared statements. |
| Threat-Level: Medium | | |

Table 16 Nikto Vulnerability 16.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| DEBUG HTTP verb may show server debugging information. See http://msdn.microsoft.com/en-us/library/e8z01xdh%28VS.80%29.aspx for details. | The is a false positive returned by a ‘200’ web error code [18]. | No mitigation required. But in order to reduce the false positive which may increase analytical expenditure through time spent analyzing false positives via Nikto, 200 error messages could be replace by outputting generic error pages instead of error codes. |
| Threat-Level: Low | | |

Table 17 Nikto Vulnerability 17.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| OSVDB-877: HTTP TRACE method is active, suggesting the host is vulnerable to XST | This is an extension of the Trace method vulnerability which allows SQL injection attacks [19]. | Again, Sanitization of user input combined with the removal of the TRACE http method from the http config file. |
| Threat-Level: Medium | | |

Table 18 Nikto Vulnerability 18.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| OSVDB-8450: /mutillidae/phpMyAdmin/db\_details\_  importdocsql.php?submit\_show=true&do=import&docpath=../:  phpMyAdmin allows directory listings remotely. Upgrade to version 2.5.3 or higher. http://www.securityfocus.com/bid/7963. | This is a windows specific CVE that allows remote attack agents to search and manipulate objects in memory which results in complete exposure of the applications databases which leads to total system integrity breach [20]. | This is mitigated by upgrading the back-end server application Apache2.0 |
| Threat-Level: High | | |

Table 19 Nikto Vulnerability 19.

|  |  |  |  |
| --- | --- | --- | --- |
| Detection | Description/Risk/Impact | | Mitigation |
| /mutillidae/index.php?page=../../../../../../../../../../etc/passwd: The PHP-Nuke Rocket add-in is vulnerable to file traversal, allowing an attacker to view any file on the host. (probably Rocket, but could be any index.php) | This vulnerability is related to the php-rocket plugin and allows an attack agent to use directory traversal to access any file on the host machine [21]. | Since this vulnerability results in potential full system loss to the attack agent, the php addon must always be kept up to date. As well as the server running the web application and the OS host. If any doubt exists regarding php-nuke, do not use it | |
| Threat-Level: High | | | |

Table 20 Nikto Vulnerability 20.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| /mutillidae/index.php?page=../../../../../../../../../../boot.ini: PHP include error may indicate local or remote file inclusion is possible. | Directory traversal giving access to the boot.ini file which could result is host system corrupt or shutdown resulting in a dos attack. | Sanitation of user input via html snippets or prepared statements. |
| Threat-Level: High |  |  |

Table 21 Nikto Vulnerability 21.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| /mutillidae/phpinfo.php?VARIABLE=<script>alert('Vulnerable')</script>: Output from the phpinfo() function was found. | This is the Deja-vu information leak vulnerability that was patched in php version 4. This results in system information leakage relating to the installed PHP version [22]. | This is mitigated by upgrading the PHP version, implementing XSS filtering and sanitizing user input. |
| Threat-Level: High |  |  |

Table 22 Nikto Vulnerability 22.

|  |  |  |  |
| --- | --- | --- | --- |
| Detection | Description/Risk/Impact | | Mitigation |
| OSVDB-12184 | This is an information leak relating to the expose\_php option being enabled [23]. | | Disbale by toggling the expose\_php option to disabled in the php.ini file found th eht elocal php folder on the server’s host machine. |
| Threat-Level: Low | | **Relates to:**  •http://209.165.200.235/ /mutillidae/?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000:  •http://209.165.200.235/mutillidae/?=PHPE9568F36-D428-11d2-A769-00AA001ACF42:  •http://209.165.200.235/mutillidae/?=PHPE9568F34-D428-11d2-A769-00AA001ACF42:  •http://209.165.200.235/ mutillidae/?=PHPE9568F35-D428-11d2-A769-00AA001ACF42: | |

Table 23 Nikto vulnerability 23.

|  |  |  |  |
| --- | --- | --- | --- |
| Detection | Description/Risk/Impact | | Mitigation |
| OSVDB-3092 | This exploit allows an attack agent to bypass authentication and alter data with direct requests. This has the potential to result in total system loss, corruption or a dos attack [24]. | | PhpMyFactures must be updated beyond version 1.2 |
| Threat-Level: Medium | | **Relates to:**   * http://209.165.200.235 /mutillidae/home/ * http://209.165.200.235/mutillidae/includes/ * http://209.165.200.235 /mutillidae/login/ * http://209.165.200.235/mutillidae/passwords/ * http://209.165.200.235/mutillidae/register/ | |

Table 24 Nikto vulnerability 24.

|  |  |  |  |
| --- | --- | --- | --- |
| Detection | Description/Risk/Impact | | Mitigation |
| OSVDB-3268 | This is a minor information leak related to page indexing | | This can be mitigated by modifying the apache2.0 config file via Options -Indexes |
| Threat-Level: Low | | **Relates to:**   * http://209.165.200.235/mutillidae/includes * http://209.165.200.235/mutillidae/passwords * http://209.165.200.235/mutillidae/images * http://209.165.200.235/mutillidae/styles * http://209.165.200.235/mutillidae/images/?pattern=etc/\*&sort=name: | |

Table 25 Nikto vulnerability 25.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| OSVDB-3233  /mutillidae/phpinfo.php: Output from the phpinfo() function was found. | This is an information leakage vulnerability which leaks information related to the program language the web application is built on and can lead to exploitation. | Firstly, keep PHP updated regularly, and secondly remove the PHPinfo page from the live website, it is not needed in production web apps. |
| Threat-Level Medium | **Relates to:**   * http://209.165.200.235/mutillidae/phpinfo.php * http://209.165.200.235/mutillidae/index.php | |

Table 26Nikto Vulnerability 26.

|  |  |  |  |
| --- | --- | --- | --- |
| Detection | Description/Risk/Impact | | Mitigation |
| OSVDB-5092 | This is an information leakage vulnerability that includes usernames and password | | This file should not contain plain text usernames and passwords. However, Access control methods can be applied to this file. A direct firewall rule can be applied prohibiting any external access to the file and host/agent-less IDS/IPS rules can be pushed to immediately severe and block any connection/IP that attempts to view this file. |
| Threat-Level: High | | **Relates to:**   * http://209.165.200.235/mutillidae/config.inc | |

Table 27 Nikto vulnerability 27.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| /mutillidae/phpinfo.php?GLOBALS[test]=<script>alert(document.cookie);</script>: Output from the phpinfo() function was found. | This is an XSS exploitation of the PHPinfo() function. | This can be mitigated by the removal of the PHPfunction() function from all pages. |
| Threat-Level: High | | |

Table 28 Nikto Vulnerability 28.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| /mutillidae/phpinfo.php?cx[]=SAMPLEINPUT<script>alert(foo)</script> | This is an XSS exploitation of the PHPinfo() function. | This can be mitigated by the removal of the PHPfunction() function from all pages. |
| Threat-Level: High | | |

Table 29 Nikto Vulnerability 29.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| 128 instances of Remote File Inclusion  Example-OSVDB-5292: /mutillidae/index.php?module=PostWrap&page=http://cirt.net/rfiinc.txt?: RFI from RSnake's list (http://ha.ckers.org/weird/rfi-locations.dat) or from http://osvdb.org/ | This is a client-side injection attack which redirects user to download a file from a remote site [25]. | All user input must be sanitized with html snippets or prepared statements. |
| Threat-Level: High | | |

Table 30 Nikto vulnerability 30.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| /mutillidae/login.php:  Admin login page/section found.  /mutillidae/phpMyAdmin/: phpMyAdmin directory found | These are admin page links that should never be readily available to but admins. | Access to admin pages should be password protected, also, link should be obfuscated by custom built selective rendering link statements in the php code which strip out unneeded text and randomize some characters in client-side output. [26]. |
| Threat Level: High | | |

Table 31 Nikto vulnerability 31

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| /mutillidae/?-s: PHP allows retrieval of the source code via the -s parameter, and may allow command execution. See http://www.kb.cert.org/vuls/id/520827  /mutillidae/login.php?-s: PHP allows retrieval of the source code via the -s parameter, and may allow command execution. See http://www.kb.cert.org/vuls/id/520827 | The ‘-s’ php switch allows for CLI variables to be passed via the UR input into the php-cgi odule which allows for the potential for arbitrary remote code exploits to be deployed [27]. | Mitigation for this vulnerability once again comes down to regular maintenance and updating of the PHP program language. |
| Threat Level: High | | |

Table 32 Nikto vulnerability 32

Nikto is very useful tool as where others can often focus on client-side exploitations, Nikto often reveals pertinent web application vulnerabilities that exist beyond the code of the web application itself yet remain in scope. Here we have seen a perfect example of Nikto’s OS and service fingerprinting revealing information that would be of crucial significance to a client if this where a real-world scan. Many of the detections are wide open to exploitation, as we would expect given that this is an OWASP™ vulnerable application; and would require immediate intervention from blue team developers.

## 3.2 OWASP™-Zap Scan Results

OWASP™-Zap Issues its reports HTML, XML and Mark Down reports depending on the user’s preference. While these are useful, they produce a significant amount of case by case vulnerability specific technical data the client simply doesn’t need to see. I would generally attach such reports as addendums to my own report and provide only that which the client needs to see in my own custom report. For the purposes of this assignment I restricted myself to only provide my custom report, though the original reports shall remain available on request for verification purposes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Detection | | Description/Risk/Impact | | Mitigation | |
| 19 instances of Possible SQL injection detected | Risk vary from client-side injection to full system compromise [28]. This method requires data to not be stored but executed directly from user input. | | | | User input should always be sanitized with methods such as http snippets or prepared statements. Coupled with this, preventative modules such as Anti-Xss, OWASP EASPI and Apache Wicket and Java/JavaScript CSRF modules; should be implemented. Cookie sessions should be restricted by the HttpOnly flag. |
| Threat-Level: High | | | **Relates to:**   * http://209.165.200.235/mutillidae/index.php?page=login.php * http://209.165.200.235/mutillidae/index.php?page=view-someones-blog.php * http://209.165.200.235/mutillidae/index.php?page=user-info.php&password=ZAP%27+AND+%271%27%3D%271%27+--+&user-info-php-submit-button=View+Account+Details&username=ZAP * http://209.165.200.235/mutillidae/index.php?page=user-info.php&password=ZAP&user-info-php-submit-button=View+Account+Details&username=ZAP%27+AND+%271%27%3D%271%27+--+ * http://209.165.200.235/mutillidae/index.php?page=login.php * http://209.165.200.235/mutillidae/index.php?page=userinfo.php&password=ZAP&user-inf * http://209.165.200.235/mutillidae/index.php?page=userinfo.php&password=ZAP&user-info-php-submitbutton=View+Account+Details+AND+1%3D1+--+&username=ZAPo-php-submit-button=View+Account+Details+AND+1%3D1+--+&username=ZA | | |

Table 33 OWASP™-Zap vulnerability 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Detection | | Description/Risk/Impact | Mitigation |
| 19 instances of Cross-Site Scripting (Reflected) | | This is a client-side injection attack caused by lack of user input sanitization coupled with the lack of preventative module implementation. The risks are varied but range from minor manipulation of client-side data, to full system compromise [28]. This method requires a non-persistent link to used or entered as input but not be stored. | User input should always be sanitized with methods such as http snippets or prepared statements. Coupled with this, preventative modules such as Anti-Xss, OWASP EASPI and Apache Wicket and Java/JavaScript CSRF modules; should be implemented. Cookie sessions should be restricted by the HttpOnly flag. |
| Threat-Level: High | **Relates to:** | | |
|  | **•**http://209.165.200.235/mutillidae/index.php?forwardurl=http%3A%2F%2Fwww.php.net%2F&page=javascript%3Aalert%281%29%3B  •http://209.165.200.235/mutillidae/index.php?page=javascript%3Aalert%281%29%3B  •http://209.165.200.235/mutillidae/index.php?page=source-viewer.php  •http://209.165.200.235/mutillidae/index.php?forwardurl=%22%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E&page=redirectandlog.php  •http://209.165.200.235/mutillidae/index.php?forwardurl=https%3A%2F%2Faddons.mozilla.org%2FenUS%2Ffirefox%2Fcollections%2Fjdruin%2Fpr%2F&page=javascript%3Aalert%281%29%3B  •http://209.165.200.235/mutillidae/index.php?page=pen-test-tool-lookup.php  •http://209.165.200.235/mutillidae/index.php?page=set-background-color.php  •http://209.165.200.235/mutillidae/index.php?choice=%3C%2Ftd%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E%3Ctd%3E&initials=ZAP&page=user-poll.php&user-poll-php-submit-button=Submit+Vote  •http://209.165.200.235/mutillidae/index.php?do=toggle-hints&page=javascript%3Aalert%281%29%3B  •http://209.165.200.235/mutillidae/index.php?page=javascript%3Aalert%281%29%3B  •http://209.165.200.235/mutillidae/index.php?choice=nmap&initials=ZAP&page=javascript%3Aalert%281%29%3B&user-poll-php-submit-button=Submit+Vote  •http://209.165.200.235/mutillidae/index.php?page=source-viewer.php  •http://209.165.200.235/mutillidae/index.php?page=add-to-your-blog.php  •http://209.165.200.235/mutillidae/index.php?page=javascript%3Aalert%281%29%3B&password=ZAP&user-info-php-submit-button=View+Account+Details&username=ZAP  •http://209.165.200.235/mutillidae/index.php?page=password-generator.php&username=%3C%2Fscript%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E%3Cscript%3E  •http://209.165.200.235/mutillidae/index.php?page=add-to-your-blog.php  •http://209.165.200.235/mutillidae/index.php?page=javascript%3Aalert%281%29%3B&username=anonymous  •http://209.165.200.235/mutillidae/index.php?page=register.php | | |

Table 34 OWASP™-Zap vulnerability 2.

|  |  |  |  |
| --- | --- | --- | --- |
| Detection | | Description/Risk/Impact | Mitigation |
| 6 instances of Cross-Site Scripting (Reflected)  (Continued from last table) | | This is a client-side injection attack caused by lack of user input sanitization coupled with the lack of preventative module implementation. The risks are varied but range from minor manipulation of client-side data, to full system compromise [28]. This method requires a non-persistent link to used or entered as input but not be stored. | User input should always be sanitized with methods such as http snippets or prepared statements. Coupled with this, preventative modules such as Anti-Xss, OWASP EASPI and Apache Wicket and Java/JavaScript CSRF modules; should be implemented. Cookie sessions should be restricted by the HttpOnly flag. |
| Threat-Level: High | **Relates to:**   * http://209.165.200.235/mutillidae/index.php?page=login.php * http://209.165.200.235/mutillidae/index.php?page=user-info.php&password=%27%22%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E&user-info-php-submit-button=View+Account+Details&username=ZAP * http://209.165.200.235/mutillidae/index.php?page=login.php * http://209.165.200.235/mutillidae/index.php?page=view-someones-blog.php * http://209.165.200.235/mutillidae/index.php?page=user-info.php&password=ZAP&user-info-php-submit-button=View+Account+Details&username=%27%22%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E * http://209.165.200.235/mutillidae/index.php?page=register.php | | |
|  |  | | |

Table 35 OWASP™-Zap vulnerability 2 (continued).

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| 10 instances of Path/Directory Traversal | Directory traversal allows attack agents to access files, webpages and carry out commands they should not have access to or permission for. Of concern are null characters and escape characters (/). Much like SQL, risk vary from client side inject to full system compromise [29]. | User input must be properly sanitized using http snippets, prepared statements and white lists/blacklists. Escape characters should be automatically screened out of any user input as should null characters. Path mapping should also be employed. All user input should be executed under the principal of least privilege. |
| Alert-Level: High | **Relates to:**   * http://209.165.200.235/mutillidae/index.php?page=source-viewer.phphttp://209.165.200.235/mutillidae/index.php * http://209.165.200.235/mutillidae/index.php?forwardurl=http%3A%2F%2Fwww.php.net%2F&page=%2Fetc%2Fpasswd * http://209.165.200.235/mutillidae/index.php?do=toggle-hints&page=%2Fetc%2Fpasswd * http://209.165.200.235/mutillidae/index.php?choice=nmap&initials=ZAP&page=%2Fetc%2Fpasswd&user-poll-php-submit-button=Submit+Vote * http://209.165.200.235/mutillidae/index.php?page=%2Fetc%2Fpasswd&password=ZAP&user-info-php-submit-button=View+Account+Details&username=ZAP * http://209.165.200.235/mutillidae/index.php?page=source-viewer.php * http://209.165.200.235/mutillidae/index.php?page=%2Fetc%2Fpasswd&username=anonymous * http://209.165.200.235/mutillidae/index.php?forwardurl=https%3A%2F%2Faddons.mozilla.org%2Fen-US%2Ffirefox%2Fcollections%2Fjdruin%2Fpr%2F&page=%2Fetc%2Fpasswd * http://209.165.200.235/mutillidae/index.php?page=%2Fetc%2Fpasswd | |

Table 36 OWASP™-Zap vulnerability 3.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| 6 instances of possible SQL injection | This is a client-side injection attack caused by lack of user input sanitization coupled with the lack of preventative module implementation. The risks are varied but range from minor manipulation of client-side data, to full system compromise [28]. This method requires a non-persistent link to used or entered as input but not be stored. | User input should always be sanitized with methods such as http snippets or prepared statements. Coupled with this, preventative modules such as Anti-Xss, OWASP EASPI and Apache Wicket and Java/JavaScript CSRF modules; should be implemented. Cookie sessions should be restricted by the HttpOnly flag. |
| Alert-Level: High | **Relates to:**   * http://209.165.200.235/mutillidae/index.php?page=user-info.php&password=ZAP&user-info-php-submit-button=View+Account+Details&username=ZAP%27+AND+%271%27%3D%271%27+--+ * http://209.165.200.235/mutillidae/index.php?page=login.php * http://209.165.200.235/mutillidae/index.php?page=view-someones-blog.php * http://209.165.200.235/mutillidae/index.php?page=user-info.php&password=ZAP%27+AND+%271%27%3D%271%27+--+&user-info-php-submit-button=View+Account+Details&username=ZAP * http://209.165.200.235/mutillidae/index.php?page=login.php * http://209.165.200.235/mutillidae/index.php?page=user-info.php&password=ZAP&user-info-php-submit-button=View+Account+Details+AND+1%3D1+--+&username=ZAP | |

Table 37 OWASP™-Zap vulnerability 4.

**Exploit Demonstration:**

Here I demonstrated several vulnerabilities together. Non persistent SQL injection, combined with information leakage, and authentication bypassing. I started off by going to the web applications login page, and simply enter one quote into the Name text input box. What I was looking for was some kind of error issued from the SQL server that might revealed the syntax of the required fields in order to execute a injection search. And I was successful.

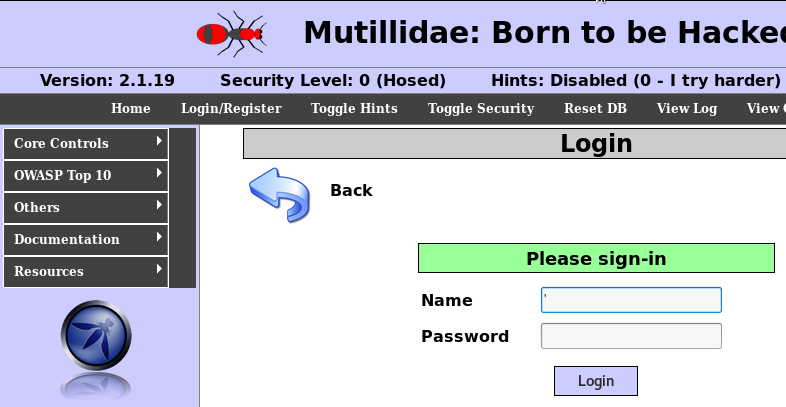


Figure 14 First i entered a single quote in an attempt to illicit an SQL database error.

And the result is:

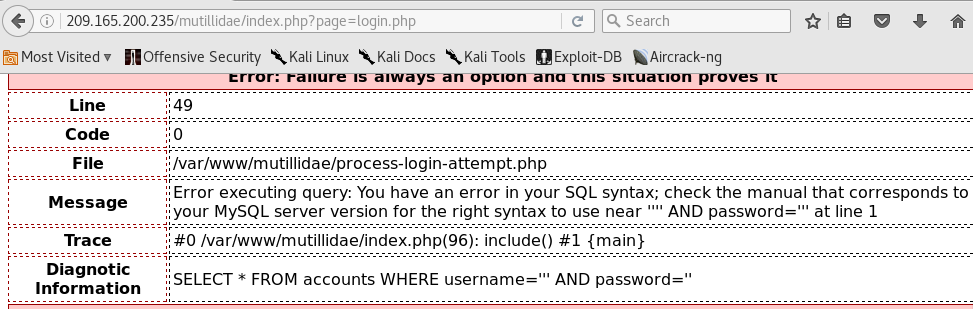


Figure 15 And here we have our desired SQL error revealed the required search syntax.

As seen above, we have forced the information leakage which reveals the required search syntax. As we can see we simply need to copy and paste it into one of the fields. However, we do not yet know a user name or password. We will bypass this, by simply using Boolean logic, as seen below.



Figure 16 Here we attempt to bypass the authentication by injecting a Boolean true statement into the hard-coded SQL search string.

As seen above we have entered the string: ‘ or 1=1 – “ . What this does is essentially tells the SQL data base to resolve to true if the user name = blank, and the password = blank OR 1=1. Naturally, 1 is always equal to 1; and statement should resolve to true.

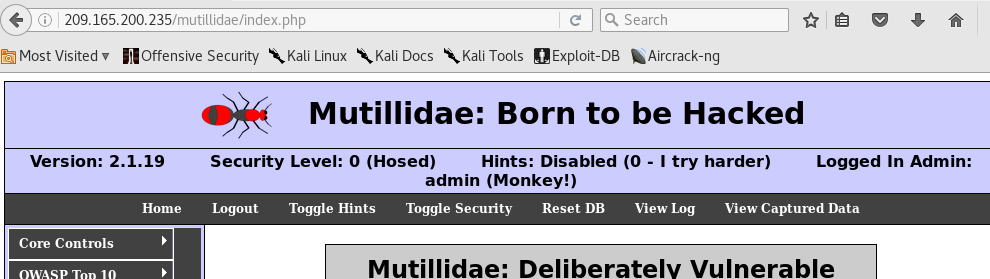


Figure 17 And we succeeded and are logged in as Admin.

And we succeeded in bypassing authentication using non persistent SQL injection with the syntax being mined from an information leak. The reason we logged in as Admin in this case is merely a co-incidence due to the first user in the data base. For this reason, it is a good practice to start your database tables from a random high number and increment from there, such as 1599. User ID’s can also be generated randomly. The first user should be hard coded to the database, without the possibility of being removed, and have no privileges. This helps preserve the database integrity.

|  |  |  |  |
| --- | --- | --- | --- |
| Detection | Description/Risk/Impact | | Mitigation |
| 1 instance of Cross-Site Scripting (Persistent) | | This is a client-side injection attack caused by lack of user input sanitization coupled with the lack of preventative module implementation. The risks are varied but range from minor manipulation of client-side data, to full system compromise [28]. This method requires that user input be accepted and stored for a period. | User input should always be sanitized with methods such as http snippets or prepared statements. Coupled with this, preventative modules such as Anti-Xss, OWASP EASPI and Apache Wicket and Java/JavaScript CSRF modules; should be implemented. Cookie sessions should be restricted by the HttpOnly flag. |
| Alert-Level: High | | **Relates to:**   * http://209.165.200.235/mutillidae/index.php?page=user-info.php&password=ZAP&user-info-php-submit-button=View+Account+Details&username=ZAP | |

Table 38 OWASP™-Zap vulnerability 5.

**Exploit Demonstration:**

I demonstrated an example of a persistent SQL injection attack by injecting a simple script into a blog user input text box. This in turn was saved to the server. When I navigate to the saved blog entry containing my script, my script was immediately executed as can be seen below.

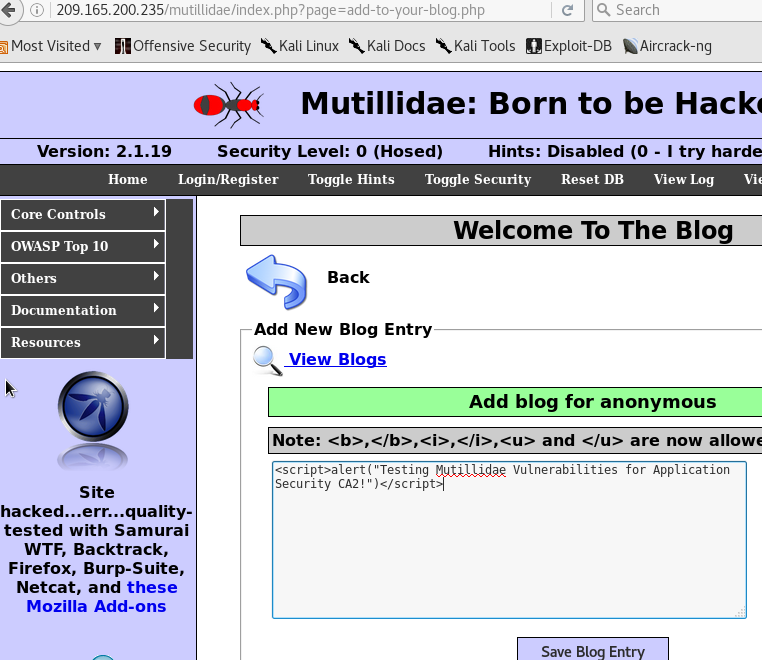


Figure 18 Here we see me entering my simple XSS script into the user input text box.

Here, I deploy my script into the user input text box.

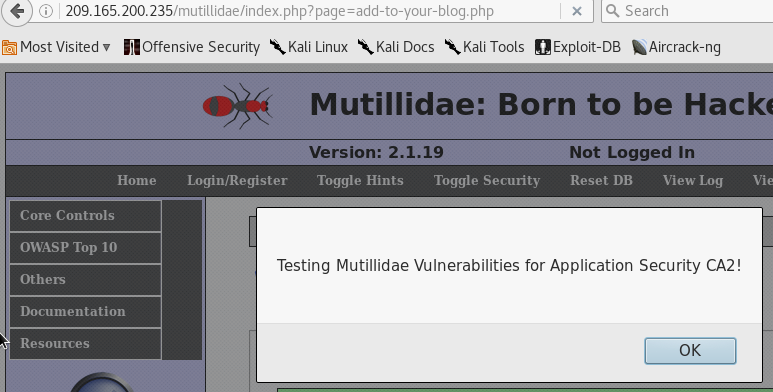


Figure 19 My script is immediately executed by the server.

The server immediately executes my code, while also saving it to the blogs database as an anonymous user. As my XSS script is now saved, it has become a persistent XSS attack, but we must still execute it from its saved state.



Figure 20 Finally, the now persistent XSS attack is launched.

And here we see the result, a persistent saved XSS attack launched from the Blog database.

|  |  |  |  |
| --- | --- | --- | --- |
| Detection | | Description/Risk/Impact | Mitigation |
| 2 instances of SQL injection may be possible | This is a client-side injection attack caused by lack of user input sanitization coupled with the lack of preventative module implementation. The risks are varied but range from minor manipulation of client-side data, to full system compromise [28]. This method does not require that user input be stored but executed directly. | | User input should always be sanitized with methods such as http snippets or prepared statements. Coupled with this, preventative modules such as Anti-XSS, OWASP EASPI and Apache Wicket and Java/JavaScript CSRF modules; should be implemented. Cookie sessions should be restricted by the HttpOnly flag. |
| Alert-Level: High | **Relates to:**   * http://209.165.200.235/mutillidae/index.php?page=pen-test-tool-lookup.php * http://209.165.200.235/mutillidae/index.php?page=view-someones-blog.php | | |

Table 39 OWASP™-Zap vulnerability 6.

|  |  |  |  |
| --- | --- | --- | --- |
| Detection | | Description/Risk/Impact | Mitigation |
| External redirect | While this may not present a risk to the system, it does present a risk to users of the system. A user could be redirected to a malicious site which could lead to the capturing of user credentials. This is turn could lead to total system compromise and database corruption or Dos attack [30]. | | User input should always be sanitized with methods such as http snippets or prepared statements. Coupled with this, preventative modules such as Anti-Xss, OWASP EASPI and Apache Wicket and Java/JavaScript CSRF modules; should be implemented. Cookie sessions should be restricted by the HttpOnly flag. Principals of least privilege should be applied to database user access and user input. Concatenated strings in input should be prohibited. |
| Alert-Level: High | **Relates to:**   * http://209.165.200.235/mutillidae/index.php?forwardurl=1098767734103624870.owasp.org&page=redirectandlog.php | | |

Table 40 OWASP™-Zap vulnerability 7.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| 102 instances of X-Frame-Options Header Not Set | X-Frame-Header is not set which allows possible Click-Jacking attacks to be injected into responses sent to users from the server. These could lead to injection attacks, CVE attacks and credential capturing attacks against users. This could result in damage to users’ machines, loss of data, and corruption of system integrity via stolen credentials [31]. | Implement the X-Frame-Header HTTP header option. This will restrict the framing of your application to pages only returned directly from the server. Accordingly, the SAMEORIGIN method should be added to potential frame screening, coupled with an implicit deny all rule. The ALLOW-FROM rule should be added to allow specific websites, which in this case means just the clients web application. |
| Alert-Level: Medium | **Relates to:**   * http://209.165.200.235/mutillidae/javascript/ddsmoothmenu/?C=N;O=D * http://209.165.200.235/mutillidae/index.php?forwardurl=http://www.room362.com/&page=redirectandlog.php * http://209.165.200.235/mutillidae/index.php?page=redirectandlog.php * http://209.165.200.235/mutillidae/index.php?page=php-errors.php * http://209.165.200.235/mutillidae/set-up-database.php * http://209.165.200.235/mutillidae/index.php?choice=nmap&initials=ZAP&page=user-poll.php&user-poll-php-submit-button=Submit+Vote * http://209.165.200.235/mutillidae/index.php?page=site-footer-xss-discussion.php * http://209.165.200.235/mutillidae/index.php?page=capture-data.php * http://209.165.200.235/mutillidae/javascript/?C=M;O=D * http://209.165.200.235/mutillidae/index.php?page=home.php * http://209.165.200.235/mutillidae/index.php?page=home.php * http://209.165.200.235/mutillidae/index.php?page=source-viewer.php * http://209.165.200.235/mutillidae/styles/?C=N;O=A * http://209.165.200.235/mutillidae/styles/ddsmoothmenu/?C=M;O=D * http://209.165.200.235/mutillidae/index.php?page=documentation/how-to-access-Mutillidae-over-Virtual-Box-network.php * http://209.165.200.235/mutillidae/index.php?page=login.php * http://209.165.200.235/mutillidae/index.php?page=user-poll.php * http://209.165.200.235/mutillidae/index.php?page=view-someones-blog.php * http://209.165.200.235/mutillidae/styles/ddsmoothmenu/?C=S;O=A * http://209.165.200.235/mutillidae/javascript/ddsmoothmenu/?C=N;O=A | |

Table 41 OWASP™-Zap vulnerability 8.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| 36 instances of Application Error Disclosure | This detection relates to default error pages revealing system related information. This information can be used by attack agents to threat hunt, and discover exploits for example, if versioning information were revealed. These alerts can also be false positives. This could result in costly analysis time or total system corruption. | Custom error pages should always be used in place of default application or source code error messages and HTTP browser default errors. Errors should be logged to a database via an encrypted connection, and never output to users. |
| Alert-Level: Medium | **Relates to:**   * http://209.165.200.235/mutillidae/styles/ddsmoothmenu/ * http://209.165.200.235/mutillidae/styles/ddsmoothmenu/?C=D;O=D * http://209.165.200.235/mutillidae/javascript/?C=S;O=D * http://209.165.200.235/mutillidae/styles/ * http://209.165.200.235/mutillidae/javascript/?C=D;O=D * http://209.165.200.235/mutillidae/javascript/ * http://209.165.200.235/mutillidae/styles/ddsmoothmenu/?C=D;O=A * http://209.165.200.235/mutillidae/styles/?C=S;O=A * http://209.165.200.235/mutillidae/javascript/?C=S;O=A * http://209.165.200.235/mutillidae/styles/?C=M;O=A * http://209.165.200.235/mutillidae/javascript/?C=N;O=D * http://209.165.200.235/mutillidae/javascript/?C=N;O=A * http://209.165.200.235/mutillidae/styles/ddsmoothmenu/?C=N;O=D * http://209.165.200.235/mutillidae/javascript/ddsmoothmenu/?C=M;O=A * http://209.165.200.235/mutillidae/styles/?C=M;O=D * http://209.165.200.235/mutillidae/styles/ddsmoothmenu/?C=S;O=A * http://209.165.200.235/mutillidae/styles/ddsmoothmenu/?C=N;O=A * http://209.165.200.235/mutillidae/javascript/ddsmoothmenu/?C=M;O=D * http://209.165.200.235/mutillidae/styles/ddsmoothmenu/?C=S;O=D * http://209.165.200.235/mutillidae/javascript/?C=D;O=A | |

Table 42 OWASP™-Zap vulnerability 9.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| 4 Instances of Directory Browsing. | The entire database of pages and folders can be enumerated and browsed at the user’s leisure [32]. | Disable directory browsing. If this is required, make sure the listed files do not induce risks. |
| Alert-Level: Medium | **Relates to:**   * http://209.165.200.235/mutillidae/styles/ * http://209.165.200.235/mutillidae/styles/ddsmoothmenu/ * http://209.165.200.235/mutillidae/javascript/ddsmoothmenu/ * http://209.165.200.235/mutillidae/javascript/ | |

Table 43 OWASP™-Zap vulnerability 10.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| 10 Instances of Parameter Tampering | Client-side manipulation of parameters via user input caused an error with output a Java stack response. This indicates a lack of exception handling. While this does not present an immediate direct risk, this information leakage could result in exploitation discovery and encourages an attack agent to hunt for more vulnerabilities. | Errors should be fixed an individual and as needed basis, however; all potential errors should be handled, and all errors messages should come in the form of default or custom created information less error page. User input should be sanitized to reduce the likelihood on errors. |
| Alert-Level: Medium | **Relates to:**   * http://209.165.200.235/mutillidae/index.php?page=%40&username=anonymous * http://209.165.200.235/mutillidae/index.php?page=%40 * http://209.165.200.235/mutillidae/index.php?forwardurl=https%3A%2F%2Faddons.mozilla.org%2Fen-US%2Ffirefox%2Fcollections%2Fjdruin%2Fpr%2F&page=%40 * http://209.165.200.235/mutillidae/index.php?page=%40 * http://209.165.200.235/mutillidae/index.php?page=source-viewer.php * http://209.165.200.235/mutillidae/index.php?do=toggle-hints&page=%40 * http://209.165.200.235/mutillidae/index.php?forwardurl=http%3A%2F%2Fwww.php.net%2F&page=%40 * http://209.165.200.235/mutillidae/index.php?choice=nmap&initials=ZAP&page=%40&user-poll-php-submit-button=Submit+Vote * http://209.165.200.235/mutillidae/index.php?page=%40&password=ZAP&user-info-php-submit-button=View+Account+Details&username=ZAP * http://209.165.200.235/mutillidae/index.php?page=source-viewer.php | |

Table 44 OWASP™-Zap vulnerability 11.

**Exploit Demonstration:**

For this exploit I used ‘Metasploit’ and ‘Armitage’ together to leverage the fact that I could inject argument into PHP functions. While the exploit is handled automatically, what we can take away from it is that the ack of user input sanitization has allowed unescaped characters to be injected. This allows the exploit to inject shell meta characters into the string. And the results can be seen below.

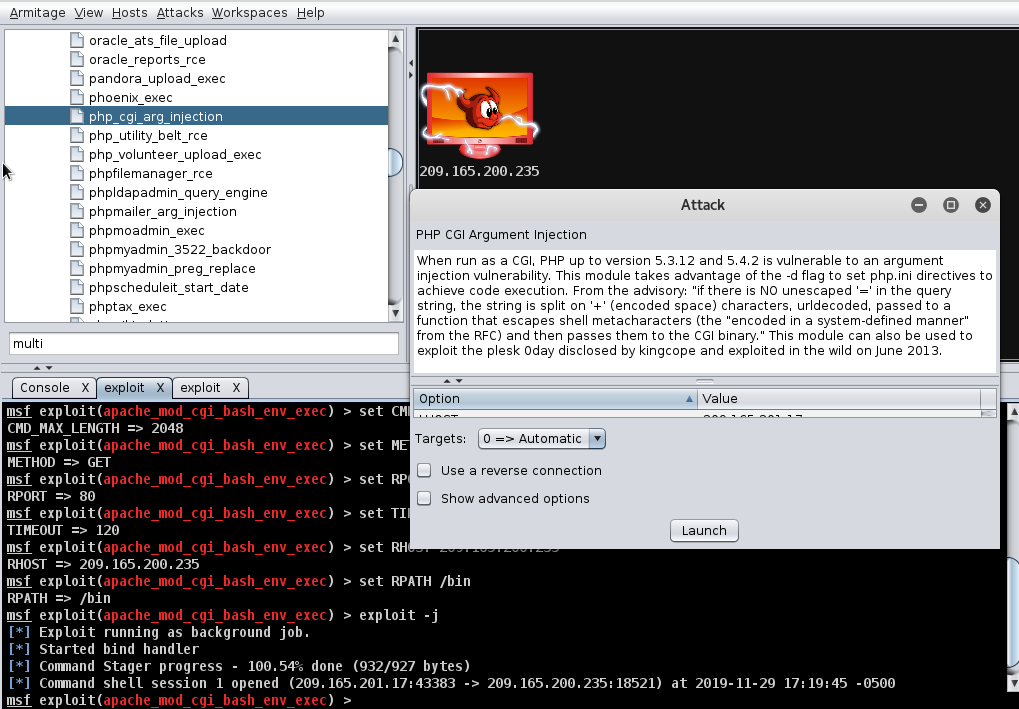


Figure 21 Here have a successful shell creation.

As seen above using the ‘php\_cgi\_arg\_injection’ exploitation leveraged through Armitage, we are return a successful shell creating message. And as seen below, we have a meterterpreter shell giving us root access to the system.

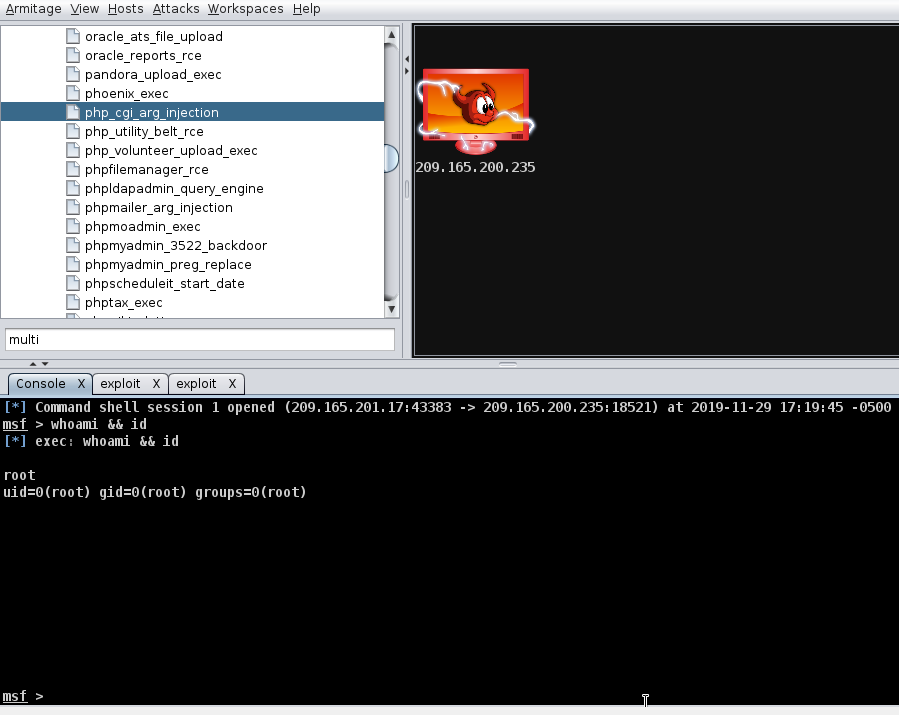


Figure 22 Root access and full system compromise.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| 154 Instances of Cookie creation without the HttpOnly Flag attached | A cookie created without the HttpOnly flag is potentially open to being intercepted, which could lead to session hijacking attacks [33]. | All cookies should be created with the HttpOnly flag attached and encrypted where ever possible. |
| Alert-Level: Low | **Relates to:**   * http://209.165.200.235/mutillidae/index.php?do=toggle-hints&page=credits.php * http://209.165.200.235/mutillidae/index.php?do=toggle-hints&page=site-footer-xss-discussion.php * http://209.165.200.235/mutillidae/ * http://209.165.200.235/mutillidae/index.php?do=toggle-security&page=pen-test-tool-lookup.php * http://209.165.200.235/mutillidae/index.php?do=toggle-hints&page=documentation/vulnerabilities.php * http://209.165.200.235/mutillidae/index.php?do=toggle-security&page=redirectandlog.php * http://209.165.200.235/mutillidae/index.php?page=text-file-viewer.php * http://209.165.200.235/mutillidae/index.php?page=password-generator.php * http://209.165.200.235/mutillidae/index.php?do=toggle-hints&page=dns-lookup.php * http://209.165.200.235/mutillidae/index.php?page=change-log.htm * http://209.165.200.235/mutillidae/index.php?do=toggle-hints&page=user-info.php * http://209.165.200.235/mutillidae/?page=add-to-your-blog.php * http://209.165.200.235/mutillidae/index.php?do=toggle-security&page=set-background-color.php * http://209.165.200.235/mutillidae/index.php?do=toggle-hints&page=site-footer-xss-discussion.php * http://209.165.200.235/mutillidae/index.php?page=php-errors.php * http://209.165.200.235/mutillidae/index.php?page=redirectandlog.php * http://209.165.200.235/mutillidae/index.php?do=toggle-hints&page=usage-instructions.php * http://209.165.200.235/mutillidae/index.php?do=toggle-hints&page=secret-administrative-pages.php * http://209.165.200.235/mutillidae/index.php?do=toggle-hints&page=documentation/how-to-access-Mutillidae-over-Virtual-Box-network.php | |

Table 45 OWASP™-Zap vulnerability 12.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| 112 Instances of X-Content-Type-Options Header Missing | This relates to miss configured Anti-MIMI-Sniffing header X-Content-Type-Options which has been set to ‘nosniff’. This allows older versions of some browsers to perform MIME-sniffing on the response body. This could allow responses to be viewed in unintended formats and types. The content type is set by modern browsers if one is declared in the code of the app, which it should be. MIME sniffing can be used to determine where cross site scripting attacks can be deployed. | Only allow updated browsers to access the web server. And ensure the file type is always declared so that it can be interpreted by modern browsers [34]. |
| Alert-Level: Low | **Relates to:**   * http://209.165.200.235/mutillidae/styles/ * http://209.165.200.235/mutillidae/styles/ddsmoothmenu/ddsmoothmenu.css * http://209.165.200.235/mutillidae/index.php?page=login.php * http://209.165.200.235/mutillidae/javascript/html5-secrets.js * http://209.165.200.235/mutillidae/index.php?page=installation.php * http://209.165.200.235/mutillidae/index.php?page=pen-test-tool-lookup.php * http://209.165.200.235/mutillidae/?page=view-someones-blog.php * http://209.165.200.235/mutillidae/?page=register.php * http://209.165.200.235/mutillidae/styles/ddsmoothmenu/?C=D;O=D * http://209.165.200.235/mutillidae/index.php * http://209.165.200.235/mutillidae/index.php?page=user-info.php * http://209.165.200.235/mutillidae/styles/ddsmoothmenu/ * http://209.165.200.235/mutillidae/index.php?page=html5-storage.php * http://209.165.200.235/mutillidae/styles/?C=S;O=A * http://209.165.200.235/mutillidae/index.php?page=source-viewer.php * http://209.165.200.235/mutillidae/index.php?page=add-to-your-blog.php * http://209.165.200.235/mutillidae/index.php?page=credits.php * http://209.165.200.235/mutillidae/index.php?page=arbitrary-file-inclusion.php * http://209.165.200.235/mutillidae/javascript/bookmark-site.js * http://209.165.200.235/mutillidae/?page=user-info.php | |

Table 46 OWASP™-Zap vulnerability 13.

|  |  |  |
| --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation |
| 104 Instances of Web Crowser XSS Protection Not Enabled | No cross-site scripting Modules have been implemented within the source code or configuration files. This leaves the web application vulnerable to XXS injection. | Cross-site scripting modules such OWASP™ EASPI Should be implemented and X-XSS-Protection should be configured to ‘1’ in the configuration file [35].  options-include:  X-XSS-Protection: 1; mode=block  X-XSS-Protection: 1; report=http://www.example.com.xss  Disable with:  X-XSS-Protection: 0  this alert is only raised if the response body could potentially contain an XSS payload (with a text-based content type, with a non-zero length). |
| Alert-Level: Low | **Relates to:**   * http://209.165.200.235/mutillidae/?page=text-file-viewer.php * http://209.165.200.235/mutillidae/index.php?page=arbitrary-file-inclusion.php * http://209.165.200.235/mutillidae/index.php?page=captured-data.php * http://209.165.200.235/mutillidae/javascript/?C=N;O=A * http://209.165.200.235/mutillidae/styles/ddsmoothmenu/?C=N;O=A * http://209.165.200.235/mutillidae/styles/?C=D;O=A * http://209.165.200.235/mutillidae/javascript/ddsmoothmenu/?C=D;O=A * http://209.165.200.235/mutillidae/index.php?page=secret-administrative-pages.php * http://209.165.200.235/mutillidae/?page=user-info.php * http://209.165.200.235/mutillidae/index.php?forwardurl=http://www.owasp.org/index.php/Louisville&page=redirectandlog.php * http://209.165.200.235/mutillidae/styles/?C=S;O=D * http://209.165.200.235/mutillidae/index.php?page=show-log.php * http://209.165.200.235/mutillidae/javascript/ddsmoothmenu/?C=S;O=A * http://209.165.200.235/mutillidae/index.php?page=documentation/vulnerabilities.php * http://209.165.200.235/mutillidae/?page=view-someones-blog.php * http://209.165.200.235/mutillidae/index.php?page=installation.php * http://209.165.200.235/mutillidae/index.php?page=view-someones-blog.php * http://209.165.200.235/mutillidae/javascript/?C=N;O=D * http://209.165.200.235/mutillidae/javascript/?C=N;O=D | |

Table 47 OWASP™-Zap vulnerability 14.

|  |  |  |  |
| --- | --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation | |
| 12 Instances of Autocomplete active in Browser | The autocomplete attribute is not disabled in HTML or configuration files. This allows users to store passwords in browsers which, while convenient presents a security risk as cookies and thus password can be hack from browsers. | | Turn off the autocomplete attribute in the forms in html or individual html input elements containing password inputs by suing ‘AUTOCOMPLETE=OFF’. |
| Alert-Level: Low | **Relates to:**   * http://209.165.200.235/mutillidae/index.php?page=user-info.php&password=ZAP&user-info-php-submit-button=View+Account+Details&username=ZAP * http://209.165.200.235/mutillidae/index.php?page=register.php * http://209.165.200.235/mutillidae/index.php?page=register.php * http://209.165.200.235/mutillidae/?page=login.php * http://209.165.200.235/mutillidae/index.php?page=login.php * http://209.165.200.235/mutillidae/?page=register.php * http://209.165.200.235/mutillidae/index.php?page=login.php * http://209.165.200.235/mutillidae/?page=register.php * http://209.165.200.235/mutillidae/index.php?page=register.php * http://209.165.200.235/mutillidae/index.php?page=user-info.php * http://209.165.200.235/mutillidae/?page=user-info.php | | |

Table 48 OWASP™-Zap vulnerability 15.

|  |  |  |  |
| --- | --- | --- | --- |
| Detection | Description/Risk/Impact | Mitigation | |
| 1 Instance of Private IP Disclosure | A private IP was discovered in the HTTP response body. This may indicate networks or servers up/down stream of the server, deeper in the client’s network that may then become the target of information scans or attacks | | Remove all mention of Ip addresses from viewable or minable sources. Instead of standard comments HTML/JavaScript comments, use JSP/ASP/PHP notations which cannot be viewed in source code. |
| Alert-Level: Low | **Relates to:**   * http://209.165.200.235/mutillidae/index.php?page=documentation/how-to-access-Mutillidae-over-Virtual-Box-network.php * 192.168.56.0, 192.168.0.0 | | |

Table 49 OWASP™-Zap vulnerability 16.

**Exploit Demonstration:**

While this may seem trivial, what we have here is all the information we need to potentially begin to pivot into the client’s network. This shows the value and power of minimal tools such as Net Discovery and Nmap. In this instance we happen to know that the IP addresses are the default gateway of the client’s machine, along with an IP address of an alternate network interface belonging to the client machines upstream IDS/IPS host machine. This machine has significant level of protections given its Snort rules and UFW rules. We can see the network, we can sniff it, we can tell that it is alive, and as such, we can begin to attack and penetrate it. It is not within the bounds of a reasonably skilled hackers’ ability to bypass such security set up. This is where Heuristic, probabilistic and deterministic rules come into effect.

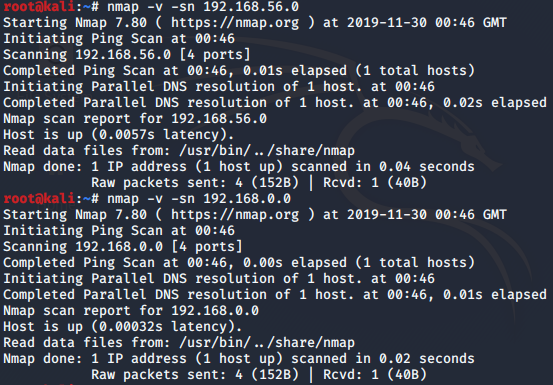


Figure 23 Nmap results from our scan of our informational leaked private IP addresses.

As we see above, our first tentative results for our first exploration upstream for the client’s web application server have resulted in two interfaces being found and detected as up and alive. This should never be allowed to happen.

# 5.0 Conclusions

While at a high level the system appears to suffer from many flaws, as we would expect, it being a purpose-built vulnerable web app; looks can be deceiving. Many of the flaws can be rectified with relatively simple adjustments to code and the implementation of common coding conventions. The addition of prepared statements, http snippets, white and black tables and anti XXS/CSRF modules would remove the vast quantity of XXS, SQL and injection-based scripts and directory traversal.

Ensuring that post blue team dev intervention based on this report, ensuring the host systems is fully up to date from the OS, the program language, the server utilities and all host-based services would further bolster the apps ability to rebuff exploitation.

I recommend the inclusion of a HIPS such as snort with full custom rules restricting access to host files, internal host addresses and any webpages that should not be accessible, and this can be complimented with local Iptables/UFW firewall rules. Coupled with the HIPS, an upstream IDS/IDS service with the inclusion of threat intelligence, such security onion running ‘ELSA’ with ‘Snort’, ‘Squil’, ‘Squert’, ‘BRO-HTML’ and all its fantastic security additions will add further security layers and can be run on a client in house server at minimal cost of upkeep and maintenance.

When comparing the results from my Nikto scans with those of OWASP™-Zap we can immediately appreciate the value of multiple scans over singular scanners. Each Scanner has its advantages and its weaknesses. Nikto adopts both a wider scanning range in terms of types of vulnerabilities screened, and a deeper level of scanning, finger printing the OS and service. While OWASP™-Zap focuses deeper into in app environment vulnerabilities and provides a much more verbose output.

The technical details from Nikto are fabulous and blue team developers would really appreciate this when it comes to implementing mitigations and fixes. While non-technical readers would love the out simple clear easily readable verbose output from OWASP™-Zap.

For myself, I greatly appreciated the ease at which I could cross reference the vulnerabilities and collate data to remove as much duplication as I felt I could remove without risking leaving vulnerabilities unpatched. I hereby present my report to be evaluated at the client’s discretion.

# 6.0 Glossary

Mutillidae: A vulnerable web application platform delivered by the OWASP™ Foundation for testing and teaching purposes.

OWASP™-Zap: A vulnerability Scanner.

Nikto: Another vulnerability scanner.

BurpSuite: A network traffic interception utility which can intercept network traffic, capture, alter, forward, decode etc. traffic between two host.

Wireshark: A packet capturing, and filtering network traffic analyzer.

Metasploitable 2.0: A Linux based vulnerable OS designed to be a teaching tool for Information security students and enthusiasts.

Metasploit: An exploitation tool aimed at scanning hosts and deliver payloads for exploitation attack vectors used by penetration tester, researchers, and hackers.

Open-Vas: A static/active code analyzer which searches for vulnerabilities in the scripting of a program or app.

Nessus: A static/active code analyzer which searches for vulnerabilities in the scripting of a program or app.

Nikto: A web app vulnerability scanner.

Wmap: A Metasploit web app analyzing module.

Edge Scan: A web app vulnerability scanner.

CMMI-DEV: Capability Maturity Model Integration-Development Framework. This is a US development and training platform with certifications being issues by the CMMI institute on behalf of the IASCA. Highly sought after by US Department of Defense contractors.

ISACA: Information System Audit and Control Association. An international IT association focused on auditing development and training platforms governing frameworks.

OS: Operating System.

OWASP™: Open Web Application Security Project (trade-marked). A highly recognized open source development and governing platform dedicated to the furtherance of web application tested and security.

MITM: Man in the Middle.

CVE: Common Vulnerability and Exposure.

ELSA: Enterprise Log Search and Archive, a network analytical and diagnostic tool used to analyze and protect networks.

SQUIL: A collection of tools which make up an implementation of a NSM.

SQUERT: Visual GUI for visualizing Squil data.

NSM: Network Security Monitoring System.

BRO: An IDS tool.

IDS: Intrusion Detection System.

CLI: Command Line Interface.

XSS: Cross Site Scripting. This is an attack inject whereby an attack agent can inject JavaScript code into user input areas of the web application that is then executed by the server.

Perl: A programming language.

Socat: Bi-directional CLI based byte stream communication utility service.

# 7.0 References

1. I.B.M. (2018, Feb), ‘Five Steps to Achieve Risk-Based Application Security Management’, 1st Edition, page 3, ‘Why you need a Risk-Based Approach to Application Security Management’, [ONILINE] Available at: <https://www.ibm.com/downloads/cas/Q4W9LP0G> [Accessed: Nov 26th, 2019]
2. B. Filkens, (2017, May) ‘Security by Design: The Role of Vulnerability Scanning in Web App Security’, 1st Editon, pages 1-3, ‘Introduction’, [ONLINE] Available at: <https://www.sans.org/reading-room/whitepapers/analyst/security-design-role-vulnerability-scanning-web-app-security-37810> \*Account Required to access pdf\* [Accessed: Nov 26th, 2019]
3. ISACA, (2017), ‘Vulnerability Assessment’, 1st Edition, page 9, ‘Vulnerability Assessment and Risk Analysis’, [ONLINE] Available at: <https://cybersecurity.isaca.org/info/cyber-aware/images/ISACA_WP_Vulnerability_Assessment_1117.pdf> [Accessed: Nov 26th, 2019]
4. Open Web Application Security Project (OWASP), ‘OWASP Mutillidae 2 Project’, OWASP™ Foundation, 2015. [ONLINE] Available at: <https://www.owasp.org/index.php/OWASP_Mutillidae_2_Project> [Accessed: Nov 26th, 2019]
5. OWASP™, ‘OWASP Zed Attack Proxy Project’ OWASP™ Foundation, Nov 26th 2019, [ONLINE] Available at: <https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project> [Accessed: Nov 26th, 2019]
6. sullo (et al) ,2019, ‘README.md’, [ONLINE] Available at: <https://github.com/sullo/nikto> [Accessed: Nov 26th, 2019]
7. B. Jackson, (2019, Aug 16th), ‘X-Frame-Options - How to Combat Clickjacking, keycdn, [ONLINE] Available at: <https://www.keycdn.com/blog/x-frame-options> [Accessed: Nov 27th, 2019]
8. B. Jackson, (2019, Jan 9th), ‘X-XSS-Protection - Preventing Cross-Site Scripting Attacks’, keycdn, [ONLINE] Available at: <https://www.keycdn.com/blog/x-xss-protection> [Accessed: Nov 27th, 2019]
9. Keycdn, (2019, Oct 4th), ‘X-Content-Type-Options HTTP Header’ [ONLINE] Available at: <https://www.keycdn.com/support/x-content-type-options> [Accessed: Nov 27th, 2019]
10. OWASP™ Foundaton, (2019, Aug 24th), ‘HttpOnly’ [ONLINE] Available at: <https://www.owasp.org/index.php/HttpOnly> [Accessed: Nov 27th, 2019]
11. Sullo (et al) , (2019, Sept 22nd) , ‘nikto/program/plugins/nikto\_headers.plugin’, line 94-101, [ONLINE] Available at: <https://github.com/sullo/nikto/blob/master/program/plugins/nikto_headers.plugin#L94-L101> [Accessed: Nov 27th, 2019]
12. Kanishka, (2017, May 30th), ‘Pentesterlab – PHP Include And Post Exploitation – Walkthrough’ ‘Let try out TCP redirection using socat’, [ONLINE] Available at: <https://medium.com/@Kan1shka9/pentesterlab-php-include-and-post-exploitation-walkthrough-8a85bcfa7b1d> [Accessed Nov, 27th 2019]
13. J Edic (et al), (2019, Sept 19th), hackerone, ‘timeline’, [ONLINE] <https://hackerone.com/reports/25382> [Accessed: Nov, 27th 2019]
14. CVE, (2014, Sept 9th), ‘CVE-2014-6271’, ‘Description’ [ONLINE] <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271> [Accessed: Nov 27th, 2019]
15. CVE, (2014, Sept 24th), ‘CVE-2014-7169’, ‘Description’ [ONLINE] <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-7169> [Accessed: Nov 27th, 2019]
16. OWASP™ Foundation, (2015, Nov 4th), ‘Test HTTP Methods (OTG-CONFIG-oo6)), ‘Test XST Potential’, [ONLINE] Available at: <https://www.owasp.org/index.php/Test_HTTP_Methods_(OTG-CONFIG-006)> [Accessed: Nov 27th, 2019]
17. Blackdoorsec, ‘Nikto’, ‘Intro’, [ONLINE] <https://www.blackdoorsec.net/archive/Nikto.php> [Accessed: Nov 27th, 2019]
18. Ethicalhack3r, (2013, Apr 22nd), ‘DEBUG Verb False Positive #79’, ‘Github’, [ONLINE] Available at: <https://github.com/sullo/nikto/issues/79> [Accessed: Nov 27th, 2019]
19. CVE, (2014, Jan 22nd), ‘CVE-2014-1636’, ‘Description’,

[ONLINE] Available at: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-1636> [Accessed: Nov 27th, 2019]

1. CVE Details, (2018, Nov 13th), ‘Vulnerability Details: CVE-2018-8450’ [ONLINE] Available at: <https://www.cvedetails.com/cve/CVE-2018-8450/> [Accessed: Nov 27th, 2019]
2. CVE Details, ‘Phpnuke >> Php-nuke : Security vulnerabilities’, [ONLINE] Available at: <https://www.cvedetails.com/vulnerability-list/vendor_id-1873/product_id-3191/Phpnuke-Php-nuke.html> [Accessed: Nov 27th, 2019)
3. Php-security, (2007, March 3rd), ‘MOPB-08-2007:PHO 4 phpinfo() XSS Vulnerability (Deja-vu)’ [ONLINE] Available at: [www.php-security.org/MOPB/MOPB-08-2007.html](http://www.php-security.org/MOPB/MOPB-08-2007.html) [Accessed Nov 27th, 2019)
4. C. Chris, (2019, Jul 30th), ‘How to enable and disable the expose\_php directive’, ‘A2 HOSTING’, [Online] Available at: <https://www.a2hosting.com/kb/developer-corner/php/using-php.ini-directives/php-expose-php-directive> [Accessed: Nov 27th, 2019]
5. CVE, (2006, June 19th), ‘CVE-2006-3092’, ‘Description’, [ONLINE] Available at: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3092> [Accessed: Nov 27th, 2019]
6. CR4Wl3R, (2010, April 18th), ‘Openscrutin 1.03 - Local File Inclusion / Remote File Inclusion’, [ONLINE] Available at: <https://www.exploit-db.com/exploits/12277> [Accessed Nov 28th, 2019]
7. T. Goldkamp, (2007), ‘ob\_start’, ‘user contributed notes’, ‘I use this to strip unnecessary characters from HTML outpit’, [ONLINE] Available at: <https://www.php.net/manual/en/function.ob-start.php#71953> [Accessed: Nov 28th, 2019]
8. Carnagie Mellon university, (2012, Dec 2nd), ‘PHP-CGI query string parameter vulnerability – Vulnerability Note VU520827’ [ONLINE] Available at: <https://www.kb.cert.org/vuls/id/520827/> [Accessed: Nov 28th, 2019]
9. CWE, (2019, Sept 19th), ‘Improper Neutralization of Input During Web Page Generation (‘Cross-site Scripting’)’, [ONLINE] Available at: <http://cwe.mitre.org/data/definitions/79.html> [Accessed: Nov 28th, 2019]
10. CWE, (2019, Sept 19th), ‘Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal'), [ONLINE] Available at: <http://cwe.mitre.org/data/definitions/22.html> [Accessed: Nov 28th, 2019]
11. CWE, (2019, June 20th), ‘URL Redirection to Untrusted Site (‘Open Redirect’)’, [ONLINE] Available at: <http://cwe.mitre.org/data/definitions/601.html> [Accessed: Nov 29th, 2019]
12. EricLaw [Edge], (2010, March 30th), ‘Combating ClickJacking With X-Frame-Options’, [Online] Available at: <https://blogs.msdn.microsoft.com/ieinternals/2010/03/30/combating-clickjacking-with-x-frame-options/> [Accessed: Nov 29th, 2019]
13. Apache, ‘Apache Core Features’,‘<Directory> Directive’, [ONLINE] Available at: <http://httpd.apache.org/docs/current/mod/core.html#options> [Accessed: Nov 29th, 2019]
14. OWASP™ Foundation, (2017, Aug 24th), ‘HttpOnly’ [ONLINE] Available at: <https://www.owasp.org/index.php/HttpOnly> [Accessed: Nov 29th, 2019]
15. Microsoft, (2016, Dec 15th), ‘Reducing MIME type security Risks’, [ONLINE] Available at: <https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/compatibility/gg622941(v=vs.85)?redirectedfrom=MSDN> [Accessed: Nov 29th, 2019]
16. OWASP™ Foundation, (2019, Nov 28th), ‘The OWASP Cheat Sheet Series was created to provide a concise collection of high value information on specific application security topics.’ [ONLINE] Available at: <https://github.com/OWASP/CheatSheetSeries> [Accessed: Nov 29th, 2019]