**MWR Internship Challenge Report**

**By Lalelani Nene**

**Username: ProfEddie**

## A. Network Scanning

### 1. Introduction

This report details the reconnaissance phase of the Capture The Flag (CTF) challenge, specifically focusing on the enumeration of Ares' mainframe server using the Nmap network scanning tool. The goal of this challenge was to gather information about the target system's operating environment, including open ports and running services, which is essential for planning further attacks.

### 2. Technical Methodology

To perform reconnaissance on Ares' mainframe server, the following steps were taken:

1. Environment Setup: A Kali Linux virtual machine was used for the scanning process. This environment is equipped with various penetration testing tools, including Nmap.
2. Initial Scan:
    - The first Nmap command executed was:

    nmap -sV -sC <IP_ADDRESS>

    - Parameters Explained:
        - -sV: This option enables version detection, which attempts to identify the version of the services running on open ports.
        - -sC: This option runs the default Nmap scripts against the discovered services to gather more information.
    - Scan Results Interpretation: After running the command, the scan results were analyzed to identify open ports and associated services.

### 3. Results

The initial scan revealed the following open ports and services on Ares' mainframe:

- Port 22:
    - Service: SSH

    - Version: OpenSSH 8.9p1 (Ubuntu)
- Port 80:
    - Service: HTTP (nginx)
    - Web Title: Test Server

This information indicated that both an SSH server and an HTTP server were operational, which can be critical for subsequent exploitation attempts.

Highest Port Identification

The highest port identified during the initial scan was 8080. However, to discover all active services, a full port scan was conducted, revealing a higher port number:

- Real Highest Port: 33333

**4. Vulnerability Analysis**

Chosen Vulnerability: Open SSH Service on Port 22

4.1 Root Cause

The SSH service running on port 22 is susceptible to various attacks, such as brute force attacks and exploitation of known vulnerabilities if not configured securely. The presence of outdated versions can further expose the system to risks.

4.2 Potential Remediations

1. Secure Configuration: Ensure that SSH is configured securely by disabling root login and using key-based authentication instead of passwords.
2. Regular Updates: Keep the OpenSSH server updated to mitigate known vulnerabilities and security flaws.
3. Intrusion Detection Systems: Implement IDS to monitor and log unauthorized access attempts.
4. Port Knocking: Use port knocking to hide the SSH port and add an additional layer of security.

**Vulnerability: OpenSSH Service on Port 22**

**Root Cause:**

The vulnerability associated with the OpenSSH service running on port 22 primarily stems from:

**Weak Configuration**: Inadequate configuration practices, such as allowing root login and reliance on password-based authentication, make the service vulnerable to brute force attacks and unauthorized access.

**Outdated Software**: Running outdated versions of OpenSSH can expose the system to known vulnerabilities and exploits that may not be mitigated in older software releases.

**Lack of Security Best Practices**: Failure to implement essential security measures, such as using key-based authentication, monitoring access logs, and setting up intrusion detection systems, further increases the risk of successful attacks.

**5. Conclusion**

The reconnaissance phase provided valuable insights into Ares' mainframe environment, which is crucial for the subsequent exploitation phase. The use of Nmap allowed for effective enumeration of open ports and services, laying the groundwork for future attacks. Addressing the vulnerabilities identified in the SSH service will enhance the system's overall security posture.

## B. Basic Steganography

**1. Introduction**

Steganography is the practice of concealing data within non-secret files to avoid detection. This technique is not merely about encryption but focuses on hiding the very existence of the message itself. This report details the exploration of steganography in digital contexts, methods to uncover hidden data, and the findings from a specific challenge involving hidden flags within a web application.

**2. Overview of Steganography**

Steganography can be applied to digital formats, including text, images, audio, and video files. Unlike cryptography, which makes messages unreadable to unauthorized parties, steganography aims to hide messages entirely. Examples include:

- **Source Code**: Programmers may embed messages in comments, variable names, or hidden links within the source code of an application.
- **Metadata**: Information that describes other data is not typically visible to users. This includes file creation dates, author details, and location coordinates, which can be manipulated to include hidden messages.
- **Covert Communications**: By embedding messages within digital files, users can communicate discreetly, avoiding detection of their interactions. However, this method can be risky, as it relies on the assumption that the hidden data will not be discovered.

**3. Techniques for Viewing Hidden Data**

Several techniques exist to extract hidden data embedded through steganography:

1. **Viewing Source Code**:
    - To access the source code of a web page, right-click on the page and select "View Page Source" or use the shortcut **Ctrl + U**. This allows examination of HTML, JavaScript, and CSS elements, which may contain hidden comments or links.
2. **Examining Metadata**:
    - **ExifTool**: A command-line utility for reading, writing, and editing metadata in various file formats.
        - To check for ExifTool, run:

            exiftool -ver
        - To install it, use:

            sudo apt install exiftool
        - To view the metadata of a file, execute:

            exiftool [path to file]

**4. Challenge Overview**

**Task Details**

The challenge involved navigating to a specified web page and uncovering hidden information regarding a killswitch flag.

**URL Accessed**:
http://MACHINE_IP/TASK2/

**Findings**

- **Common Tool for Viewing Metadata**: The tool used to view file metadata is **ExifTool**.
- **Hidden Information in Source Code**: A secret web page located in the application source code is:
    - /ares-possible-hq-locations
- **Hidden Killswitch Flag**: Upon navigating to the secret web page, the hidden killswitch flag identified was:
    - MWR{Secret-Server-Is-Under-MtEverest}

**Vulnerability: Steganography in Digital Contexts**

**Root Cause:**
The root cause of vulnerabilities associated with steganography in digital contexts stems from:

- **Concealment Techniques:** The very nature of steganography—hiding data within non-secret files—creates opportunities for misuse. This allows unauthorized individuals to embed and communicate hidden messages without detection, leading to potential security breaches.
- **Lack of Detection Mechanisms:** The absence of effective monitoring and detection tools makes it challenging to identify hidden data within digital files. Traditional security measures often overlook steganography, which can allow malicious activities to go unnoticed.
- **Manipulation of Metadata:** Steganographic methods exploit file metadata, which is typically unmonitored and can be easily manipulated to include covert messages. This exploitation increases the risk of hidden data being used for nefarious purposes.
- **Inadequate Awareness:** A general lack of awareness about steganography techniques among users and administrators can lead to insufficient security measures, making systems vulnerable to attacks leveraging hidden communications.

**5. Conclusion**

Steganography serves as a powerful tool for hiding information within ordinary files, and techniques like examining source code and metadata are crucial for uncovering such hidden messages. The challenge demonstrated practical applications of these techniques, leading to the successful retrieval of a hidden killswitch flag.

## C. Basic Cryptography

**1. Introduction**

Cryptography and encoding are essential practices in securing information and facilitating data transmission across various platforms. This report covers the fundamental differences between the two concepts, specific encoding methods like Base64, and simple encryption techniques such as ROT13.

**2. Cryptography**

Cryptography is the practice of securing information and communications by transforming readable data (plaintext) into an unreadable format (ciphertext) using various techniques. Key aspects include:

- **Encryption**: The process of converting plaintext into ciphertext using a key.
- **Decryption**: The process of converting ciphertext back into plaintext using the corresponding key.

Common algorithms in use today include the Advanced Encryption Standard (AES), which ensures data confidentiality.

**3. Encoding**

Encoding refers to the transformation of data from one format to another for purposes such as storage, transmission, or data transformation. Unlike encryption, the goal of encoding is not to keep information secret but to ensure that it can be properly processed by different systems.

**4. Base64 Encoding**

Base64 is a method for encoding binary data into an ASCII string format, which is useful for transmitting data over text-based systems, such as email. The process involves:

- Dividing binary data into 6-bit chunks.

- Mapping each chunk to a corresponding Base64 character using a predefined table.
- Padding the resulting string with = characters to ensure its length is a multiple of 4.

**5. Challenge Overview**

**Task Details**

The challenge involved decoding messages encoded in Base64 and ROT13, as well as recovering hidden information from a data fragment.

**Findings:**

Base64 Decoding:

**Encoded text:** VGhlIGtleSBsaWVzIHdpdGhpbiB0aGUgZGlnaXRhbCBzaGFkb3dz

**Decoded text:** "The key lies within the digital shadows"

ROT13 Decryption:

Encrypted text: Hapbire gur uvqqra gehgu

Decrypted text: "Uncover the hidden truth"

Final Data Fragment Recovery:

**Encoded fragment**: WkpFe05lcmYtdW5mLW4teHZ5eWZqdmdwdX0=

**Decoded text**: "MWR{Ares-has-a-killswitch}"

**Vulnerability: Basic Cryptography**

**Root Cause:**

The root cause of vulnerabilities in basic cryptography arises from several factors:

- **Weak Encryption Algorithms:** The use of outdated or insecure encryption algorithms, such as simple techniques like ROT13, can make data easily susceptible

to attacks. Attackers can exploit these weaknesses to decrypt sensitive information without proper authorization.

- **Key Management Issues:** Inadequate management of encryption keys, such as poor key generation practices, lack of key rotation, or failure to securely store keys, can lead to unauthorized access to encrypted data. If keys are compromised, the confidentiality of the data is at risk.
- **Misunderstanding Between Cryptography and Encoding:** A lack of clarity regarding the difference between cryptography and encoding can lead to improper implementation. For instance, relying on encoding methods like Base64 for security instead of using proper encryption can give a false sense of security.
- **Implementation Flaws:** Errors in the implementation of cryptographic algorithms can introduce vulnerabilities, such as improper padding, incorrect key usage, or failure to apply strong security practices during encryption and decryption processes.
- **Insufficient Security Awareness:** A general lack of knowledge regarding cryptographic principles and practices among users and developers can result in poor security decisions, leading to vulnerabilities in systems and applications.

### 6. Conclusion

Understanding cryptography and encoding techniques is vital in modern digital communications. The successful decoding of Base64 and ROT13 messages highlights the practical application of these concepts. The final data fragment recovered offers critical insight into the ongoing challenges faced in the Digital Abyss.

## D. Basic SQL Injection

### 1. Introduction

SQL Injection (SQLi) is a prevalent security vulnerability that allows attackers to manipulate the queries an application makes to its database. This report discusses the fundamentals of SQL Injection, its causes, the exploitation of a basic SQLi vulnerability, and the challenge faced in bypassing a login page to access sensitive data.

## Understanding SQL Injection

### Definition

SQL Injection occurs when an application improperly processes user input, allowing attackers to alter the SQL commands sent to the database. This vulnerability can lead to unauthorized access, data breaches, data loss, and complete control over the database server.

### Causes of SQL Injection

SQLi is primarily caused by the following:

- Dynamic SQL query construction without proper input validation or parameterization.
- Direct concatenation or insecure interpolation of user input into SQL commands.

## Exploiting Basic SQL Injection

### Overview of the Challenge

The challenge involved exploiting a SQLi vulnerability in a web application to bypass its login page and gain access to the Ares killswitch. The application's login mechanism constructed a SQL query based on user inputs for username and password.

### Analyzing the SQL Query

When attempting to log in with the username admin and the password a, the executed query looked like this:

SELECT * FROM users WHERE username = 'admin' AND password = 'hashed_password';

The password was stored as a hash in the database, which obscured the actual password value.

### Identifying the Injection Point

The SQL query's WHERE clause was identified as the point of injection. To bypass authentication, the goal was to manipulate the query so that it would evaluate to TRUE regardless of the actual username and password.

### Constructing the Payload

1. **Initial Injection:** To break the query structure, an attacker could input:

This results in an unmatched single quote, indicating a potential vulnerability.

2. **Exploiting the Logic:** To ensure the query evaluates to TRUE, an attacker could append:

OR 1=1

Thus, the payload becomes:

' OR 1=1

3. **Commenting Out the Remaining Query:** To prevent the unmatched quote from causing an error, the payload was finalized by adding a SQL comment:

' OR 1=1 -- -

**Final Query Result**

Inputting the final payload into the username field along with any password (e.g., a) modified the SQL query to:

SELECT * FROM users WHERE username = '' OR 1=1 -- - AND password = 'a';

This query bypasses the authentication mechanism by ensuring that the condition always evaluates to TRUE, granting access to the first user in the database.

**Vulnerability: SQL Injection (SQLi)**

**Root Cause:**
The root cause of SQL Injection vulnerabilities is the improper handling of user input in web applications. Specifically, this occurs when applications construct SQL queries dynamically without proper validation or sanitization of user inputs. This lack of input validation allows attackers to manipulate SQL queries by injecting malicious SQL code. Common practices that contribute to this vulnerability include:

- **Dynamic SQL Query Construction:** Developers often concatenate user input directly into SQL statements, which can lead to unintended execution of injected SQL commands.
- **Failure to Use Prepared Statements:** Not implementing prepared statements or parameterized queries allows for direct execution of injected input without safeguards, increasing the risk of SQL injection.

**Conclusion**

The challenge successfully demonstrated the exploitation of a SQL injection vulnerability, enabling unauthorized access to the application. Understanding SQLi is critical for developers and security professionals, as it highlights the need for secure coding practices, including input validation and the use of prepared statements.

## E. Logic Flaws

**Introduction**

Logic flaws are vulnerabilities that arise when a system's intended functionality is misinterpreted or inadequately implemented. This report explores the nature of logic flaws, their causes, and methods for exploitation, culminating in a specific challenge involving the exploitation of a logic flaw in an e-commerce application.

**Understanding Logic Flaws**

**Definition**

Logic flaws occur when the actual behavior of an application diverges from its intended design. These flaws can lead to unexpected outcomes and potentially compromise system security. In the context of the Ares domain, the presence of such flaws offers an opportunity to exploit its defenses.

**Causes of Logic Flaws**

The root causes of logic flaws typically include:

1. **Misunderstanding of Requirements:** Developers may misinterpret business logic or requirements, leading to incorrect implementation.
2. **Inadequate Handling of Edge Cases:** Failure to consider all possible user interactions or edge cases can result in vulnerabilities.
3. **Incorrect Assumptions About User Behavior:** Developers may make assumptions about how users will interact with the application, which may not reflect reality.
4. **Lack of Thorough Testing:** Insufficient testing for unusual scenarios can leave logic flaws unaddressed.

**Exploiting Logic Flaws**

Exploiting logic flaws requires a systematic approach to identify discrepancies between the intended and actual behavior of an application. The following steps outline a general methodology:

1. **Analyze Application Workflows:** Examine the application's processes, focusing on critical decision points and data flows.
2. **Identify Assumptions:** Look for areas where the application makes incorrect assumptions about user actions or data.
3. **Test Edge Cases:** Experiment with unusual inputs and bypass normal workflows to observe how the application responds.
4. **Manipulate User Actions:** Exploit the identified logic flaws to achieve unintended behaviors.

**Challenge: Exploiting the Logic Flaw in the E-commerce Application**

**Overview of the Challenge**

In this specific challenge, the task was to navigate to a designated URL within the e-commerce application and exploit a logic flaw within the cart discount coupon functionality. The goal was to purchase an OmniKey by manipulating this flaw.

**Steps Taken**

1. **Navigate to the Challenge URL:** The URL provided was accessed to enter the application and begin the exploration.
2. **Identify the Logic Flaw:** The application's discount coupon functionality was analyzed. It was determined that there was a logic flaw that allowed for unintended discount application during the purchasing process.
3. **Exploit the Flaw:** By manipulating the discount coupon input or the cart workflow, the flaw was exploited to receive an undeserved discount on the OmniKey.
4. **Complete the Purchase:** After successfully exploiting the logic flaw, the OmniKey was added to the cart, and the purchase was completed.
5. **Retrieving the Key's Secret:** Upon purchasing the OmniKey, the secret key was retrieved and entered as part of the challenge solution.

**Vulnerability: Logic Flaws**

**Root Cause:**
The root causes of logic flaws stem from several key issues in the software development process:

1. **Misunderstanding of Requirements:** Developers may misinterpret business logic or user requirements, leading to incorrect implementation of features. This misunderstanding can result in logic flaws that deviate from the intended functionality.
2. **Inadequate Handling of Edge Cases:** When developers fail to consider all possible user interactions or edge cases, it can lead to vulnerabilities. Applications may not respond correctly to unexpected inputs, creating opportunities for exploitation.
3. **Incorrect Assumptions About User Behavior:** Developers often make assumptions regarding how users will interact with the application. If these assumptions are flawed, it can lead to scenarios where users can exploit the application in ways not anticipated by the developers.
4. **Lack of Thorough Testing:** Insufficient testing for unusual scenarios or edge cases can leave logic flaws unaddressed. Comprehensive testing is essential to identify and resolve these vulnerabilities before deployment.
5. **Inadequate Documentation and Communication:** Poor documentation or communication between stakeholders can exacerbate misunderstandings about the application's intended behavior, leading to the introduction of logic flaws.

**Conclusion**

Successfully exploiting the logic flaw within the e-commerce application demonstrated the importance of understanding both the application's intended functionality and the potential gaps in its implementation. Logic flaws can serve as critical vulnerabilities if left unaddressed.

Participating in the MWR internship challenge has provided valuable insights into various aspects of cybersecurity, including reconnaissance, steganography, cryptography, and SQL injection. These experiences will undoubtedly contribute to my growth as a cybersecurity professional and prepare me for future challenges in the field.