Pen Test Questionnaire

Offensive Security Consultant

Network Scanning and Enumeration:

Review the nmap commands below.
What is the purpose of these commands, and what potential information can you gather from its output?

- `nmap -p- -Pn [target IP]`
- Answer box

> The -p- option scans all TCP ports (65535) on the target IP, -Pn skips the discovery phase by disabling the ICMP ping echo request. This is useful if the host does not respond to this request, but it is still discoverable. A list would be provided with open, closed, and filtered ports. Open ports accept connections while closed do not and filtered ports are when nmap is unsure such as from firewall or filtered packets that could filter the ports.

What is the purpose of this nmap command?

- `nmap -sS -sV -O -p- --script=default,vuln 192.168.1.0/24` Answer
- box

> The -sS option sends a TCP SYN packet to hosts in the 192.168.1.0/24 network. -sS doesn't complete the full TCP handshake resulting in a stealthier scan that is harder to detect by IDS and firewalls. -sV option provides version of open ports, -O provides operating system information. All TCP ports are scanned with -p- option. –script=default,vuln option is an nmap script that aims to detect vulnerabilities on the hosts.

What is the purpose of this nmap command?

- `nmap -sS -f --data-length 200 192.168.1.105`
- Answer box

> The 192.168.1.105 IP address is scanned using smaller SYN packet fragments to avoid detection by IDS or if the host is blocking certain types of packets. 200 byte data is sent in every packet using –data-length which can reveal how the host reacts to handling unexpected or malformed packets.

What is the purpose of this nmap command?

```
nmap -6 -sT -p 22,80,443 fe80::20c:29ff:fe21:5771
```
Answer box

The -6 option specifies nmap to scan an IPv6 addresses as the host is an IPv6 address. The -sT option performs a full TCP scan to the ports 22 (SSH), 80 (HTTP) and 443 (HTTPS)

Wireshark Packet Analysis:

Given the below packet capture snippet from Wireshark, can you deduce if this is part of a normal connection setup or a potential scanning activity?

Explain your reasoning based on the packet details.

```
No.   Time     Source     Destination  Protocol Length Info
1   0.000000  10.0.0.1   192.168.1.105 TCP    74   34567 → 80 [SYN] Seq=0 Win=65535 Len=0
    MSS=1460
2   0.001234  192.168.1.105 10.0.0.1   TCP    74   80 → 34567 [SYN, ACK] Seq=0 Ack=1
Win=65535 Len=0 MSS=1460
3   0.002345  10.0.0.1   192.168.1.105 TCP    66   34567 → 80 [ACK] Seq=1 Ack=1 Win=65535
    Len=0
```

Answer box

Could be both as the host 10.0.0.1 performs a 3-way TCP handshake with the destination 192.168.1.105 which is a normal TCP setup. This can also be done on Nmap using the -sT command. More details would be required to identify the type of activity such as analysing the overall traffic pattern.

SSL/TLS handshake:

Given the captured TLS handshake, what can you infer about the client's security posture based on the cipher suites offered?

```
Secure Sockets Layer
  TLSv1.2 Record Layer: Handshake Protocol: Client Hello
Version: TLS 1.2 (0x0303)     Random: ...
    Cipher Suites Length: 58
    Cipher Suites (29 suites)
         Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
      ...
```

Answer box

The client has a strong security posture as the client offers two cipher suites within TLS 1.2 with strong encryption as they use the AES algorithm with 256 and 128 bit key lengths. They also use ECDHE for forward secrecy which ensures that past communications can't be decrypted even if the private key is compromised. Both GCM and SHA provide confidentiality and integrity and authenticity.

Low Level Code Analysis

Analyse the provided C code. Identify the vulnerability present and suggest how it can be exploited. Also, propose a mitigation strategy.

```c
#include <stdio.h>
#include <string.h>

void vulnerableFunction(char *input) {
              char    buffer[50];
strcpy(buffer, input);
}

int      main(int      argc,      char      *argv[])      {
vulnerableFunction(argv[1]);
  return 0;
}
```

Answer box

The vulnerability within this code is within the vulnerableFunction function within strcpy which is used to copy the contents of the input into a 50byte buffer. If the input exceeds 50 bytes then the memory would be overwritten causing a buffer overflow. This can be exploited by inserting an input string longer than 50 bytes. This can be mitigated by using input validation to ensure the length does not exceed the buffer size. Using alternative functions such as strncpy or snprintf should be used as they copy the specific number of bytes to prevent a buffer overflow.

Cryptographic Flaws:

Assess the cryptographic key generation function in the snippet. What makes it insecure and how would you improve it?

```python
import random

def                    weak_key_generation():
random.seed(12345) # Weak seeding
    key = ''.join([chr(random.randint(0, 255)) for i in range(16)])
return key
```

```
print("Weak cryptographic key:", weak_key_generation())
```
Answer

box

The random.seed function uses a fixed seed which is predictable and susceptible to attacks. Not only this, the number 12345 is commonly used and can be guessed. The random module in python is cryptographically insecure as it uses a pseudo-random number generator which doesn't provide sufficient randomness for cryptography. A 16 byte key length is insufficient for modern applications. To improve the function, a cryptographically secure module should be used to generate random integers securely. A stronger seed can be used, and the key length could be increased to 32 bytes for higher security against brute force attacks.

Cryptographic Algorithm Analysis:

Analyse the cryptographic code snippet. Identify any weaknesses in the encryption process and discuss potential attack vectors. Suggest improvements to the encryption scheme.

```
from Crypto.Cipher import AES
import os

def encrypt_data(data):
    key = os.urandom(16)  # 128-bit key
cipher = AES.new(key, AES.MODE_ECB)
   padded_data = data + (16 - len(data) % 16) * chr(16 - len(data) % 16)
return cipher.encrypt(padded_data)

encrypted = encrypt_data("Sensitive Data")
```
Answer

box

ECB mode is used which is a weakness as the same plaintext block is encrypted to the same cyphertext block making it vulnerable to attacks such as pattern recognition and ciphertext manipulation. The attack vectors can be from a known plain text attack if the attacker identifies patterns in the ciphertext and understand information about the plaintext. A chosen plaintext attack can recover the encryption key from observing the ciphertext. To prevent these vulnerabilities, ECB mode can be switched to CBC or GCM which provide better security properties.

Web Application Security

Examine the Python code snippet below. Identify any security vulnerabilities and discuss how they can be exploited. Also, recommend secure coding practices to fix these vulnerabilities.

```
import sqlite3

def check_credentials(username, password):
      conn  =  sqlite3.connect('users.db')
cursor = conn.cursor()
  query = f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'"
cursor.execute(query)     return cursor.fetchone()
 username = input("Username: ") password
= input("Password: ") if
check_credentials(username, password):
    print("Login successful!")
else:             print("Login
failed.")
```

Answer box

The query variable is concatenated making it vulnerable to an SQL injection attack. If the attacker inputs ‘ ’ OR 1=1 --' as the password, then the query would bypass the password check. Best practices are to parameterise the username and password. Input validation and sanitisation can be implemented for the parameters for them to conform to expected formats and to not contain malicious characters.

HTML/Javascript Code Analysis

Review the HTML and JavaScript code. Identify any vulnerabilities and explain how they can be exploited. Suggest a mitigation strategy to prevent such attacks.

```
<html> <body>
<form action="/search">
  <input type="text" name="query">
  <input type="submit" value="Search">
</form>
<p>Search Results for: <span id="searchTerm"></span></p>
<script>
document.getElementById('searchTerm').innerText = new
URLSearchParams(window.location.search).get('query'); </script>
</body> </html>
```

Answer box

The code is vulnerable to a XSS attack as the query parameter from the URL is directly assigned to innerText in span element with the id searchTerm so whatever value

passed as the query parameter in the URL is inserted into the HTML content of the page without any sanitisation or validation. An XSS attack can performed by inserting malicious code within the query. This can be mitigated by sanitising and validating user input by applying HTML encoding to the query parameter before inserting it into the HTML content to prevent it from being interpreted as code and the user input to conform to specific formats and lengths.

PHP Code

Evaluate the PHP code for potential issues. What are the weaknesses? and how might they be exploited? Suggest improvements to strengthen the authentication process.

```
<?php session_start();

if (isset($_POST['username']) && isset($_POST['password'])) {
    $username = $_POST['username'];
    $password = $_POST['password'];

    if (login($username, $password)) {
$_SESSION['user'] = $username;
echo "Login successful";
    } else {
        echo "Login failed";
    }
}

function login($username, $password) {
    $conn = new mysqli('localhost', 'db_user', 'db_password', 'db_name');
    // Check for database connection error
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }

    $sql = "SELECT * FROM users WHERE username = '$username' AND password =
'$password'";
    $result = $conn->query($sql);
if ($result->num_rows > 0) {
// login successful
return true;     } else {
        // login failed
return false;
    }
    $conn->close();
}
?>
```

Answer box

The SQL queries are constructed from user input in $username and $password that are concatenated in the SQL query string making the code vulnerable to SQL injection attacks. This can be mitigated by using prepared statements. Passwords are stored in plain text without encryption making them susceptible to be obtained by an attacker. This can be avoided by encrypting and hashing the password. The PHP session could benefit from management practices by including session expiration and regeneration to prevent session related attacks.

Python Code

Inspect the Python code for vulnerabilities. Identify any vulnerabilities and explain how an attacker could exploit them.

Recommend secure coding practices to mitigate this risk.

```
import os

def ping_host(host):
  response = os.system(f"ping -c 1 {host}")
if response == 0:          return "Host
is reachable"  else:
    return "Host is unreachable"

host = input("Enter a host to ping: ")
print(ping_host(host))
```

Answer box

The host variable defined by the user is incorporated into the os.system function which allows an attacker to manipulate the input to execute system commands. This can be mitigated through input sanitisation and validation, avoiding using os.system and similar functions that execute shell commands.

Python Flask Code

Analyse the provided code snippet. Identify any vulnerabilities found. Suggest mitigation strategies to prevent such security issues.

```
from flask import Flask, request, render_template_string j
app = Flask(__name__)

@app.route('/greet',    methods=['GET'])    def    greet():
user_name = request.args.get('name', 'Guest')    greeting =
render_template_string(f"Hello, {user_name}!")      return
greeting
     if     __name__     ==
"__main__":
  app.run(debug=True)
```

Answer box

A vulnerability is found in the render_template_string function as it evaluates the input string as a template which can lead to a server-side template injection. This can be mitigated by avoiding the function and replacing it with a function containing static template files such as render_template which prevents user input for the template code.

SQL Injection

Analyse the SQL query. Explain how this query can be used in a SQL injection attack. Discuss the use of obfuscation techniques in SQL injection and how you would bypass common security filters.

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

Answer box

The query selects all columns from the users table as the username is empty from the ‘ ‘ command or it returns all rows from the users table from the 1=1 command as it is always true thus bypassing the username authentication check. The – denotes an SQL comment making the database ignore the rest of the query so the password is not checked.

XSS Bypass

Evaluate the JavaScript code for XSS vulnerabilities, considering the filtering in place. How would you craft an XSS payload to bypass the filter and execute malicious JavaScript?

```
var userInput = document.getElementById('userInput').value;
userInput = userInput.replace(/script/gi, "");
document.getElementById('output').innerHTML = userInput;
```

Answer box

This code attempts to mitigate XSS vulnerabilities by filtering out the work script in the user input before displaying it on the webpage however, it doesn’t prevent all XSS attacks. To bypass the filter, the word script can be used in different casing or encoding, event handlers can be used to without using the word script and characters can be encoded using HTML entities.

Incident Response and Analysis:

Evaluate the bash script provided. What type of information is it designed to collect and how can this information be used in a post-exploitation phase of a pentest?

```
#!/bin/bash echo "Gathering System Information..." uname -a ifconfig ps -aux netstat -tuln
```

Answer box

This script collects system information from the target system by beginning with displaying Gathering System Information from echo command, the uname -a command retrieves information about the system kernel and OS, ifconfig displays the network interface information, ps -aux lists information about all processes running on the system including their ID’s, memory usage and names and the netstat -tuln provides active network connections, listening ports and associated processes. This helps to understand the system more and potentially find more vulnerabilities.

.