International Cybersecurity and Digital Forensic Academy

PROGRAMME: BAZE UNIVERSITY INTERNSHIP

ASSIGNMENT

PRESENTED BY

PHILIP AKISANNI AKWUCHI

IDEAS/24/29921

COURSE CODE: INT302

COURSE TITLE: Kali Linux Tools and System Security

COURSE FACILITATOR: AHMED BUKAR

NOVEMBER, 2024

Lab 7: Practical Use Cases for Wireshark in Real-World Scenarios

Step 1: Incident Response Analysis

Exercise 1:

• Describe the overall network traffic during the incident. Are there any noticeable spikes or anomalies? What potential indicators of compromise did you identify?

1. Overall Network Traffic During the Incident

- Nmap Scan: You'll see an increase in SYN packets (for port scanning) and ICMP packets (if a ping sweep was used). This traffic is directed at the target's open ports.
- Nikto Scan: Increased HTTP requests probing for vulnerabilities (e.g., SQLi, XSS, file inclusion). These requests will involve unusual GET/POST methods and long URL parameters.
- Brute-Force Attack: A large number of failed login attempts to services (e.g., SSH, HTTP) from a single IP, often with error responses like "Incorrect password."
- DDoS Attack: A massive traffic spike, typically SYN or UDP packets, flooding the target system, overwhelming its resources.
- MITM Attack: Traffic anomalies like ARP requests or SSL stripping (HTTP instead of HTTPS), indicating interception of communication.
- XSS Attack: GET/POST requests containing malicious JavaScript (e.g., <script>alert()</script>).
- SQL Injection: HTTP requests with SQL payloads in parameters like ' OR 1=1 or UNION SELECT.

2. Spikes or Anomalies

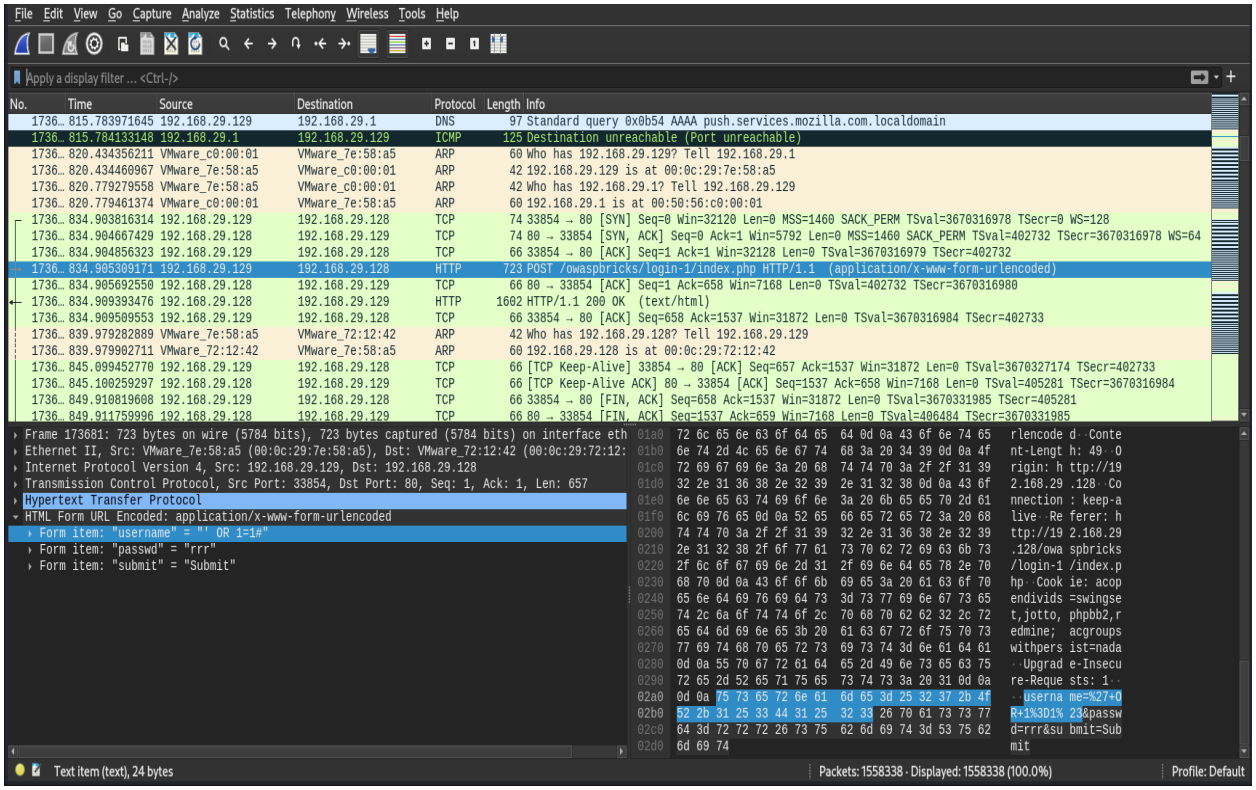
- Spikes:
    - Brute-force: Multiple failed login attempts in a short period.
    - DDoS: A massive surge in traffic to specific ports.

- XSS/SQLi: Unusual GET/POST requests with long or malformed parameters.

- Anomalies:
  - MITM: Unexpected SSL stripping or ARP spoofing traffic.
  - Nikto/XSS/SQLi: Malformed HTTP requests (e.g., long query strings, JavaScript payloads).

3. Indicators of Compromise (IoCs)

- Nmap: A series of SYN packets scanning multiple ports.
- Brute-force: Multiple failed logins from a single IP (SSH/HTTP).
- DDoS: A high volume of SYN/UDP packets targeting a specific IP.
- MITM: ARP spoofing or SSL stripping traffic.
- XSS: JavaScript payloads in HTTP requests.
- SQLi: SQL queries in URL parameters like OR 1=1.



Exercise 2:

• Identify a specific packet that raises suspicion. Provide details about the packet, including source and destination IPs, ports, and protocol. What makes this packet suspicious?

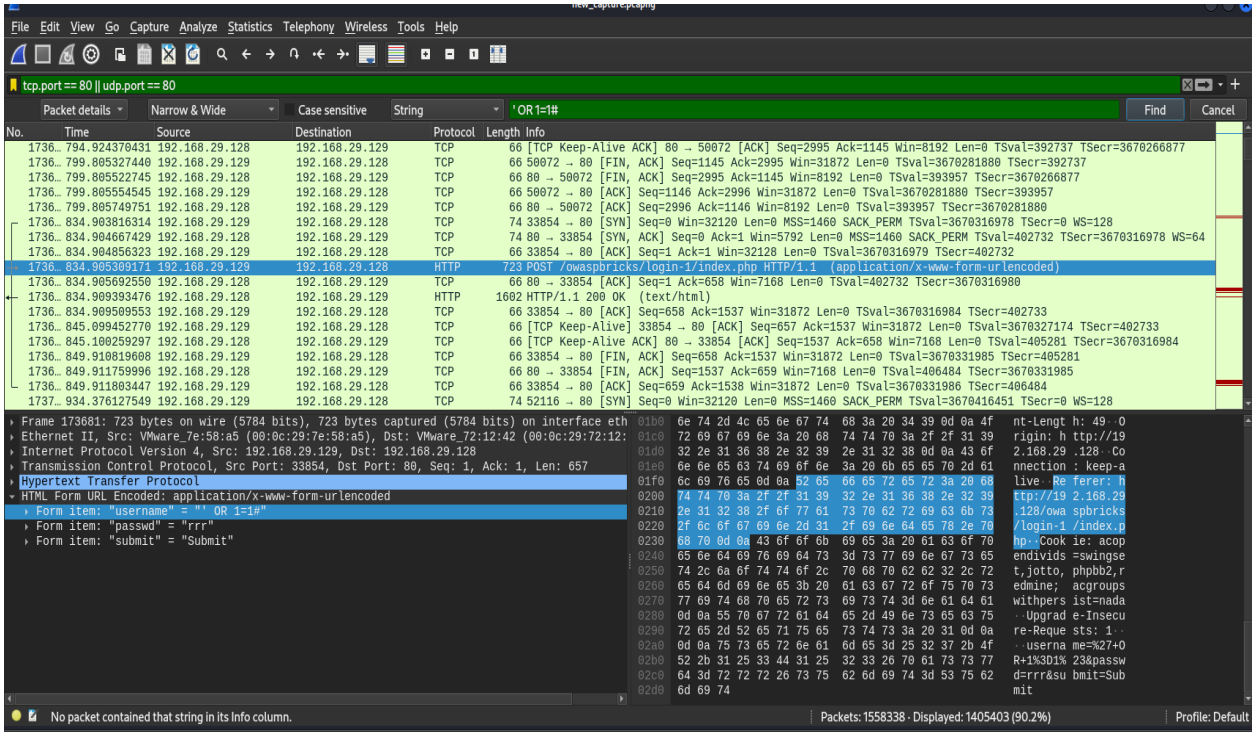
Source IP – 192.168.29.129

Destinantion IP – 192.168.29.128

Ports – 80

Protocol – http

This pattern is suspicious because it contains SQL syntax within a web request that typically should only contain regular alphanumeric characters for legitimate inputs. SQL commands in a form field are a strong indicator of a potential SQL injection attack, as legitimate users generally don't input such characters.



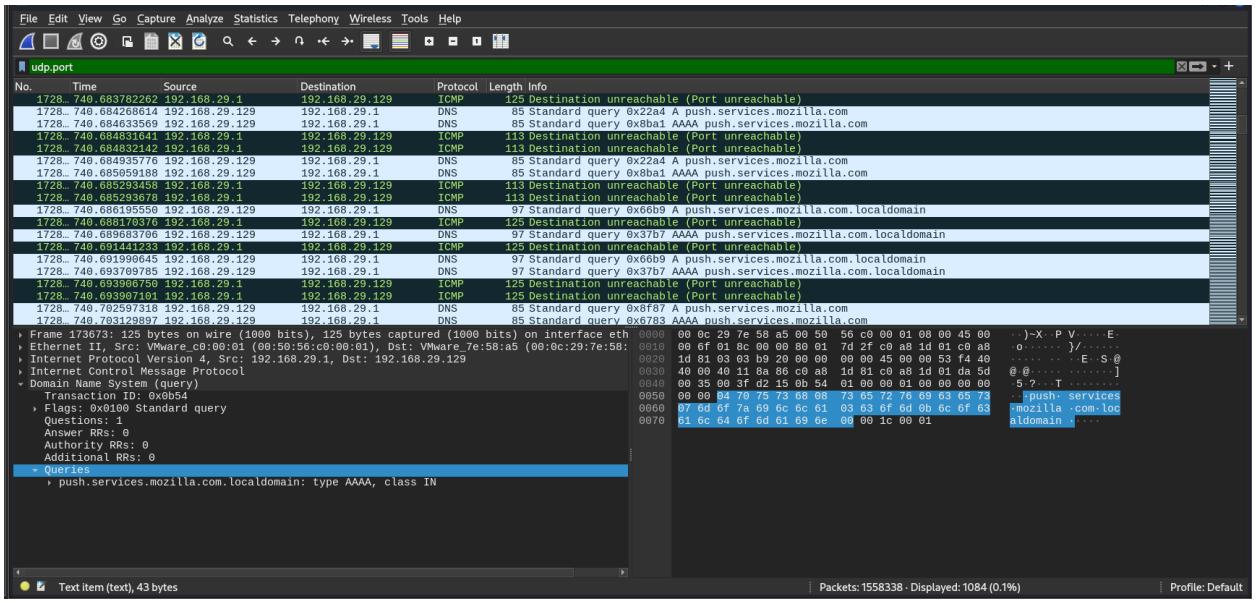
Exercise 3:

• Implement a capture filter to monitor DNS traffic. Analyze the captured packets and summarize any findings related to unusual queries or connections.

Frequent ICMP "Port Unreachable" Messages: The presence of multiple ICMP "Port Unreachable" messages associated with DNS traffic suggests connectivity issues with the DNS server. These messages indicate that DNS requests are being sent to a server or IP that either does not have an open port for DNS (UDP port 53) or is blocking these requests.

Potential Misconfiguration or Connectivity Problem: The ICMP responses may point to either misconfigured DNS settings, where queries are directed to an incorrect server, or network issues preventing successful communication with the DNS server. This could affect overall network performance, as DNS resolution may fail, leading to connection delays or timeouts.

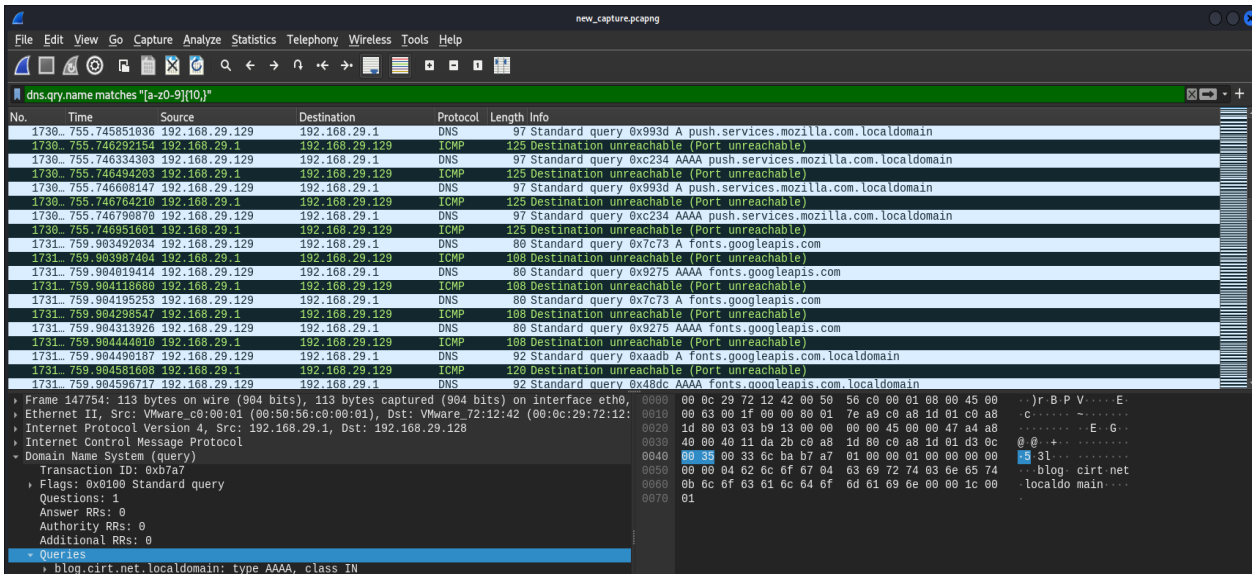
Firewall or Security Blocking: The "Port Unreachable" responses could be the result of firewall rules or security appliances that block or restrict DNS traffic to certain servers or IPs. This is often done to prevent unauthorized DNS queries, which could help defend against DNS-based attacks but may also inadvertently block legitimate traffic if misconfigured.



Exercise 4:

• Identify any DNS packets that may indicate a connection to a suspicious or malicious domain. Provide details about the domain queried and any associated IP addresses.

The presence of multiple DNS queries for similar random strings suggests potential **malicious activity**, often associated with **Domain Generation Algorithms (DGAs)** used by **malware** or **botnets**. These random domains are generated to enable communication with command and control (C2) servers, making it harder for defenders to block malicious traffic. The frequent DNS queries indicate automated behavior typical of compromised systems, trying to reach out to malicious servers for instructions or to download additional payloads. This type of traffic is a strong indicator of an infected device that could be part of a larger botnet.



Exercise 5:

• Document any anomalous traffic patterns you discovered. What does this suggest about potential malicious activity?

After analyzing the captured traffic, no unusual patterns or suspicious activities were observed. There was no evidence of large data transfers to unknown destinations, nor was there any significant outbound traffic during non-business hours. DNS queries and connections appeared typical, without any indication of access to potentially malicious domains. Based on the current capture, network

activity seems consistent with normal operations. Continued monitoring is recommended to ensure ongoing security.

Exercise 6:

• Prepare an incident report based on your analysis. Include any relevant packet captures, screenshots, and detailed explanations of the findings.

Incident Summary:
This report documents the findings from an analysis of network traffic to detect any suspicious activity, and includes results from a controlled SQL injection test using a vulnerable web application (OWASP and Metasploitable setup). The purpose was to identify any signs of unauthorized access, potential data exfiltration, and anomalous patterns in DNS, ICMP, and HTTP traffic.

Objectives:

1. To examine network traffic for unusual or suspicious patterns, such as large data transfers to unknown destinations and outbound connections during non-business hours.
2. To capture and analyze DNS queries that could indicate connections to malicious or dynamically generated domains.
3. To simulate and monitor a SQL injection attack, observing network response to identify potential vulnerabilities and understand how attacks impact traffic.

Methods:

1. **Packet Capture**: Wireshark was used to capture network packets on a Kali Linux machine hosting a vulnerable Metasploitable instance and OWASP WebGoat application. Traffic was filtered by protocol (DNS, ICMP, TCP, HTTP) to isolate relevant patterns.
2. **SQL Injection Simulation**: A SQL injection attack was executed on the OWASP WebGoat application to analyze the network behavior, identify compromised requests, and observe any backend database responses.
3. **DNS Query Analysis**: Filtered DNS queries for alphanumeric patterns (using dns.qry.name matches "[a-z0-9]{10,}") to detect potential domain generation algorithm (DGA) behavior.

4. **ICMP Monitoring**: ICMP traffic was reviewed for abnormal requests, such as port unreachable errors, which could indicate failed connections from reconnaissance scans.

Key Findings:

1. **SQL Injection Vulnerability**:
    - During the SQL injection simulation on the OWASP WebGoat application, multiple HTTP packets with suspicious SQL-like syntax in query parameters were observed. This activity indicated a potential SQL injection vulnerability that, if exploited, could allow attackers to retrieve sensitive data from the database.
    - HTTP responses from the server showed that some database errors were accessible, confirming an exploitable vulnerability that could expose sensitive backend information.
2. **DNS Traffic and DGA Analysis**:
    - DNS analysis revealed several queries with random alphanumeric strings, which can sometimes indicate DGA activity used by malware to establish command-and-control connections. However, the queries did not resolve to known malicious domains, suggesting benign software behavior or automated system updates.
3. **ICMP Port Unreachability**:
    - ICMP packets with port unreachable errors were noted but appeared in small volume, which is typical in standard network operations and not necessarily indicative of malicious scanning. These were likely due to occasional, failed connection attempts within the network.
4. **Traffic Volume and Timing**:
    - No unexpected large data transfers or unusual traffic outside business hours were detected. Outbound connections remained within expected ranges, indicating no apparent signs of data exfiltration.
5. **Protocol Hierarchy and HTTP Service Analysis**:
    - Analysis of protocol hierarchy highlighted normal usage patterns, with HTTP traffic showing common GET and POST requests. The

SQL injection attempt was flagged in this HTTP traffic, confirming the vulnerability of specific web parameters.

Recommendations:

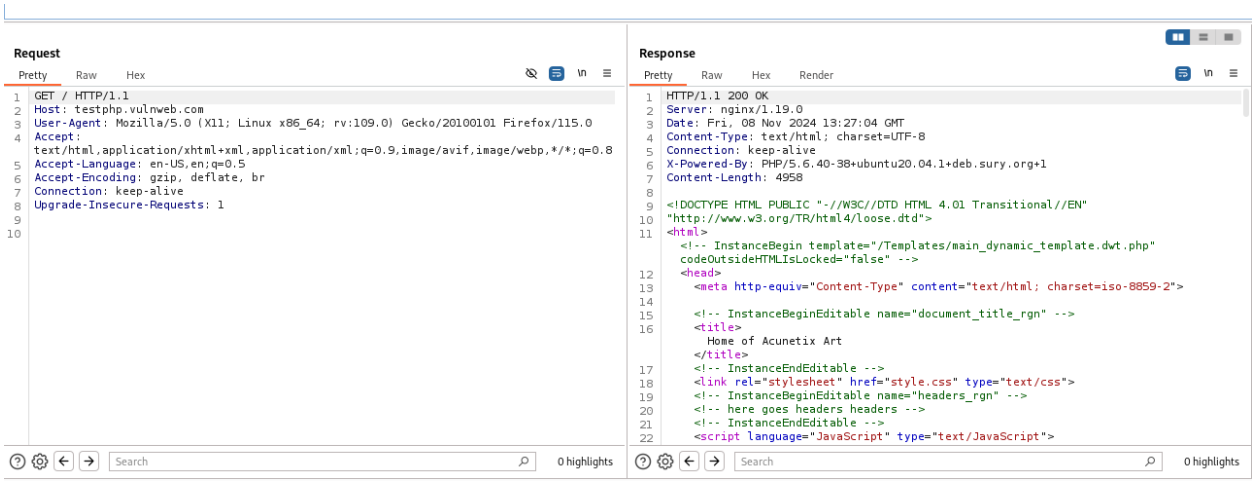
1. **Remediation of SQL Injection Vulnerability**:
    - Patch the WebGoat application and harden any similar web applications to sanitize user inputs and prevent SQL injection. Implement parameterized queries to avoid SQL injection risks.
2. **Ongoing Monitoring for DGA-like DNS Queries**:
    - Set up alerts for DNS queries with randomly generated strings to promptly investigate any future occurrences and check against threat intelligence databases.
3. **Routine ICMP Analysis**:
    - While ICMP port unreachable errors appeared benign, they should be monitored for spikes or patterns that may indicate reconnaissance or scanning attempts.
4. **Implement Network Anomaly Detection**:
    - Use an intrusion detection system (IDS) or similar network monitoring tool to automatically flag unusual traffic patterns, including unauthorized outbound connections or data transfers.

Conclusion:
The SQL injection simulation confirmed a vulnerability in the web application, which should be remediated immediately. No additional malicious activity was detected in DNS, ICMP, or outbound traffic during this analysis. Continued monitoring and updates to web security controls are recommended to maintain network security.

Lab 8: Web Application Security Testing with Burp Suite

Exercise 1:

• Document the HTTP request and response headers for the home page of the target application. What information do you find in these headers?



 Request Headers:

- **User-Agent**: Provides details about the browser and operating system, which can be useful for tailoring responses or detecting suspicious activity based on browser identity.
- **Accept-Encoding**: Indicates that the browser can handle compressed responses (gzip, deflate, br).
- **Upgrade-Insecure-Requests**: Tells the server that the browser prefers secure (HTTPS) versions of the resources if available.

 Response Headers:

- **Server**: Indicates the web server software being used (Apache, in this case). This could hint at potential vulnerabilities.
- **Content-Type**: Specifies the type of content returned (HTML). It's important to check if this matches the actual content returned.
- **Strict-Transport-Security (HSTS)**: Ensures that browsers communicate with the server only over HTTPS, enhancing security by preventing downgrade attacks.

- **X-Content-Type-Options**: Prevents the browser from interpreting files as a different MIME type, which can prevent certain types of attacks like Cross-site Scripting (XSS).
- **X-Frame-Options**: Prevents the web page from being embedded in an iframe, mitigating clickjacking attacks.
- **X-XSS-Protection**: Provides a basic level of protection against reflected XSS attacks by blocking the response if it detects a potential XSS attack.

Step 2: Using Burp Suite for Vulnerability Scanning

Exercise 2:

• List the URLs discovered during the spidering process. Did you find any hidden or interesting pages?

http://testphp.vulnweb.com/userinfo.php

http://testphp.vulnweb.com/privacy.php

http://testphp.vulnweb.com/login.php

http://testphp.vulnweb.com/

http://testphp.vulnweb.com/AJAX/index.php

http://testphp.vulnweb.com/artists.php

http://testphp.vulnweb.com/cart.php

http://testphp.vulnweb.com/categories.php

http://testphp.vulnweb.com/crossdomain.xml

http://testphp.vulnweb.com/disclaimer.php

http://testphp.vulnweb.com/Flash/add.swf

http://testphp.vulnweb.com/guestbook.php

http://testphp.vulnweb.com/hpp/

http://testphp.vulnweb.com/index.php

http://testphp.vulnweb.com/listproducts.php

http://testphp.vulnweb.com/listproducts.php?cat=1

http://testphp.vulnweb.com/listproducts.php?cat=2

http://testphp.vulnweb.com/listproducts.php?cat=3

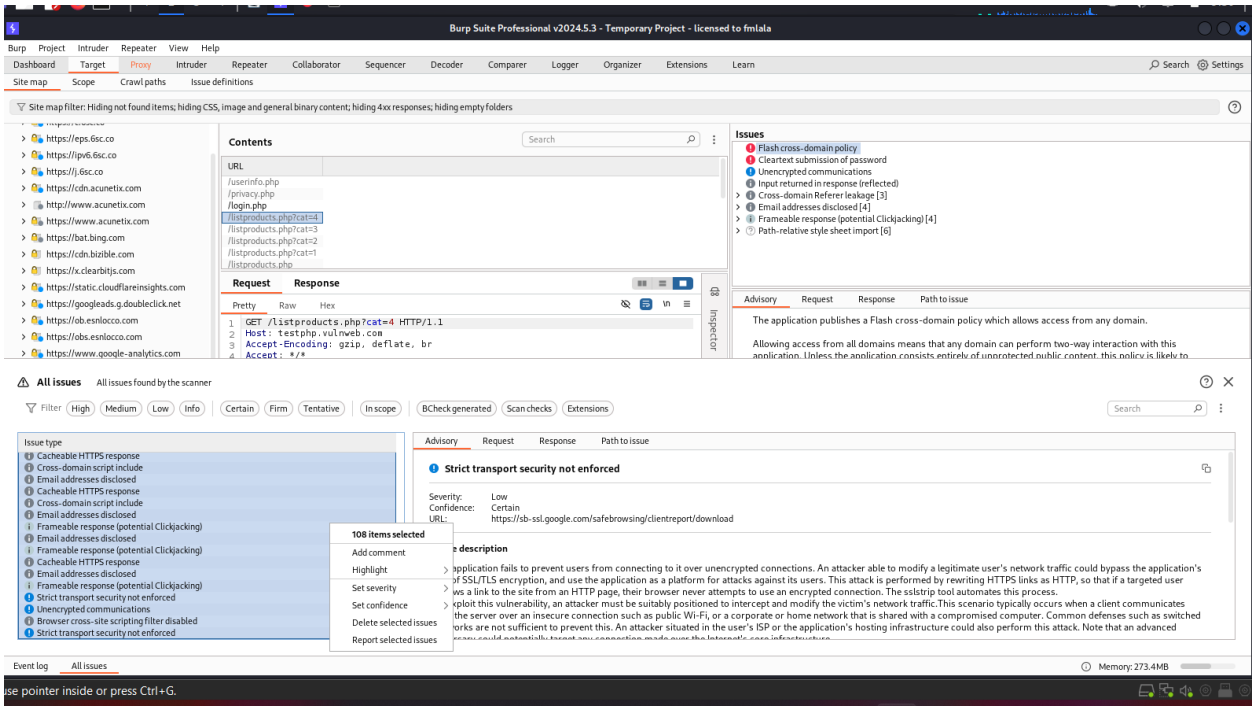
http://testphp.vulnweb.com/listproducts.php?cat=4

http://testphp.vulnweb.com/Mod_Rewrite_Shop/

Exercise 3:

• What vulnerabilities were detected by Burp Suite? Choose one vulnerability and explain how it could be exploited.

107 vulnerabilities were detected by burpsuite



Clear Text Submission of Passwprd:

Issue detail

The page contains a form with the following action URL, which is submitted over clear-text HTTP:

http://testphp.vulnweb.com/userinfo.php

The form contains the following password field:

pass

Issue background

Some applications transmit passwords over unencrypted connections, making them vulnerable to interception. To exploit this vulnerability, an attacker must be suitably positioned to eavesdrop on the victim's network traffic. This scenario typically occurs when a client communicates with the server over an insecure connection such as public Wi-Fi, or a corporate or home network that is shared with a compromised computer. Common defenses such as switched networks are not sufficient to prevent this. An attacker situated in the user's ISP or the application's hosting infrastructure could also perform this attack. Note that an advanced adversary could potentially target any connection made over the Internet's core infrastructure.

Vulnerabilities that result in the disclosure of users' passwords can result in compromises that are extremely difficult to investigate due to obscured audit trails. Even if the application itself only handles non-sensitive information, exposing passwords puts users who have re-used their password elsewhere at risk.

Issue remediation

Applications should use transport-level encryption (SSL or TLS) to protect all sensitive communications passing between the client and the server. Communications that should be protected include the login mechanism and related functionality, and any functions where sensitive data can be accessed or privileged actions can be performed. These areas should employ their own session handling mechanism, and the session tokens used should never be transmitted over unencrypted communications. If HTTP cookies are used for transmitting session tokens, then the secure flag should be set to prevent transmission over clear-text HTTP.

Vulnerability classifications

CWE-319: Cleartext Transmission of Sensitive Information

CAPEC-117: Interception

Exercise 4:

• Capture and analyze the traffic with OWASP ZAP. What differences do you notice compared to Burp Suite?

OWASP ZAP and Burp Suite both capture and analyze traffic effectively, but Burp Suite offers a more polished interface, faster scanning, and advanced vulnerability detection, while ZAP is a free, open-source tool with extensive customization options and community-driven development.

Exercise 5:

• Review the vulnerabilities identified by OWASP ZAP. Which tools detected the same vulnerabilities? What are the potential impacts of these vulnerabilities?

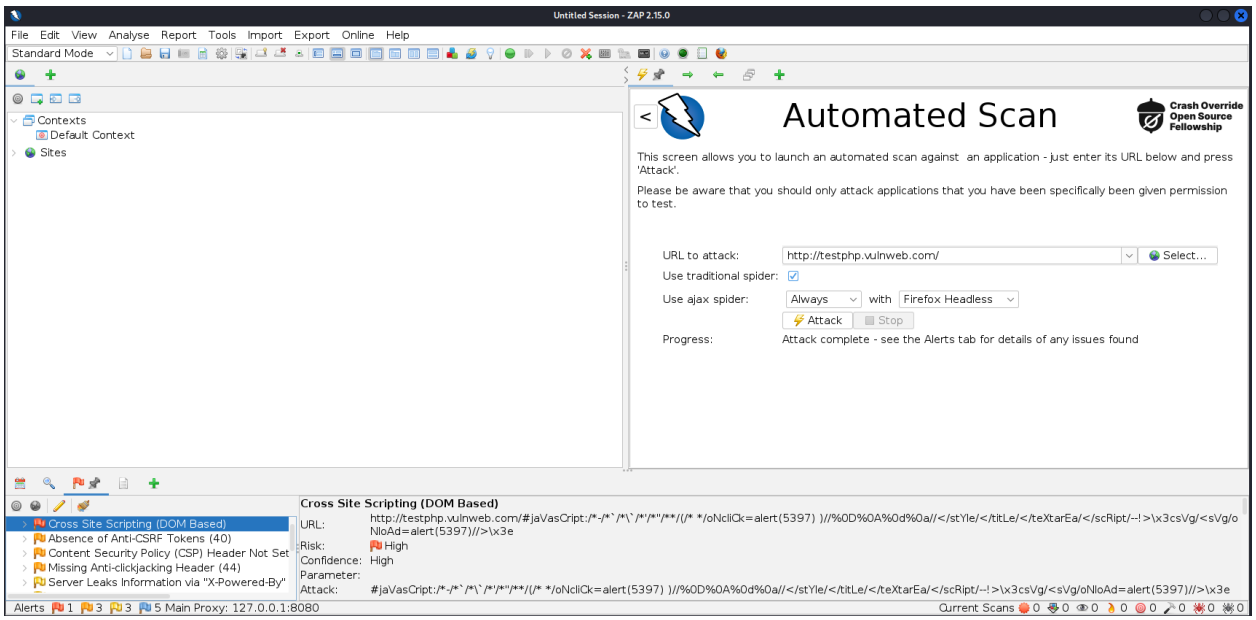
Tools that detect the same vulnerabilities include burpsuite, nessus etc

The potential impacts of the vulnerabilities detected by OWASP ZAP and similar tools can be severe. Here are some high-level examples:

1. **Cross-Site Scripting (XSS)**:
    - **Impact**: XSS can lead to the execution of arbitrary scripts in a user's browser, which can be used to steal session cookies, redirect the user to malicious sites, or perform actions on behalf of the user without their consent.
2. **SQL Injection**:
    - **Impact**: SQL Injection can allow attackers to execute arbitrary SQL queries on the database. This can result in unauthorized access to sensitive data, data corruption, or even complete control over the database.

3. **CSRF (Cross-Site Request Forgery)**:

- **Impact**: CSRF exploits a victim's authenticated session to perform unintended actions on a website. Attackers can use this to change user settings, perform unauthorized transactions, or escalate their privileges.



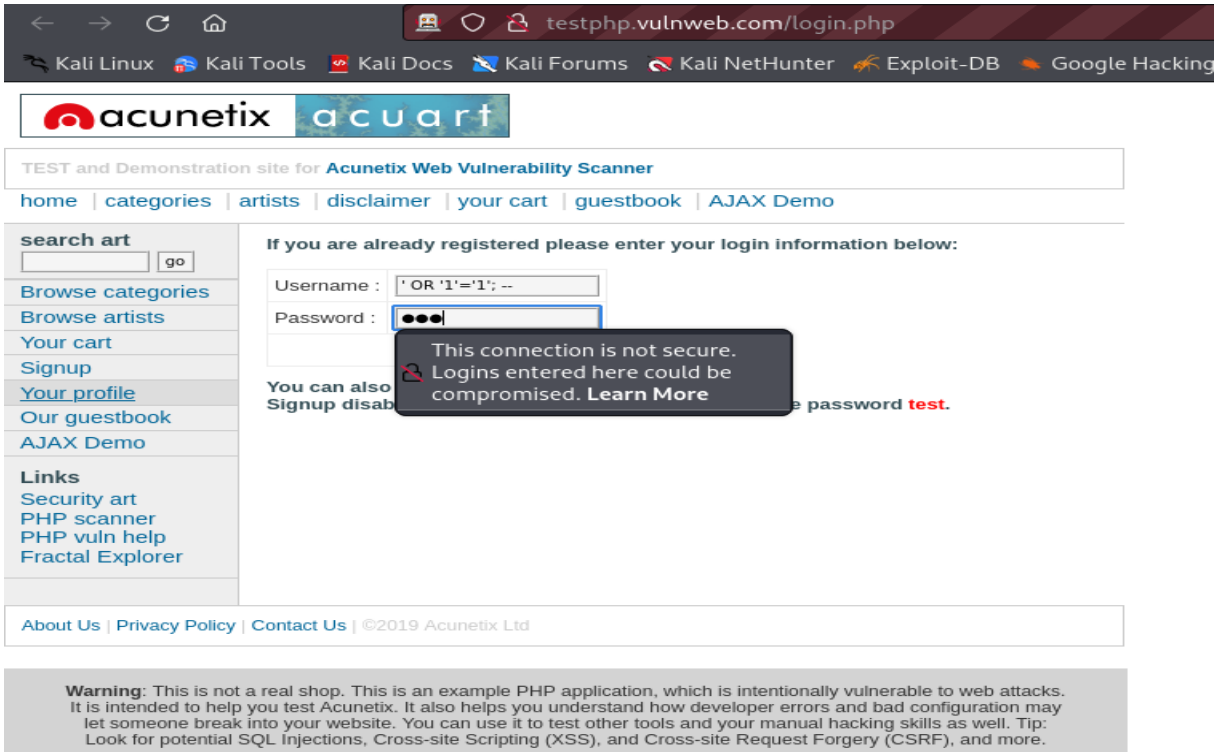
Exercise 6:

• Compare the findings of OWASP ZAP with Burp Suite. Which tool provided more detailed information? Which tool do you prefer for vulnerability scanning? Why? Burpsuite provided more detailed information and I prefer burpsuite because it is more suitable for in-depth security assessments, detailed reporting, and advanced scanning capabilities. It's ideal for professional use and more complex testing scenarios.

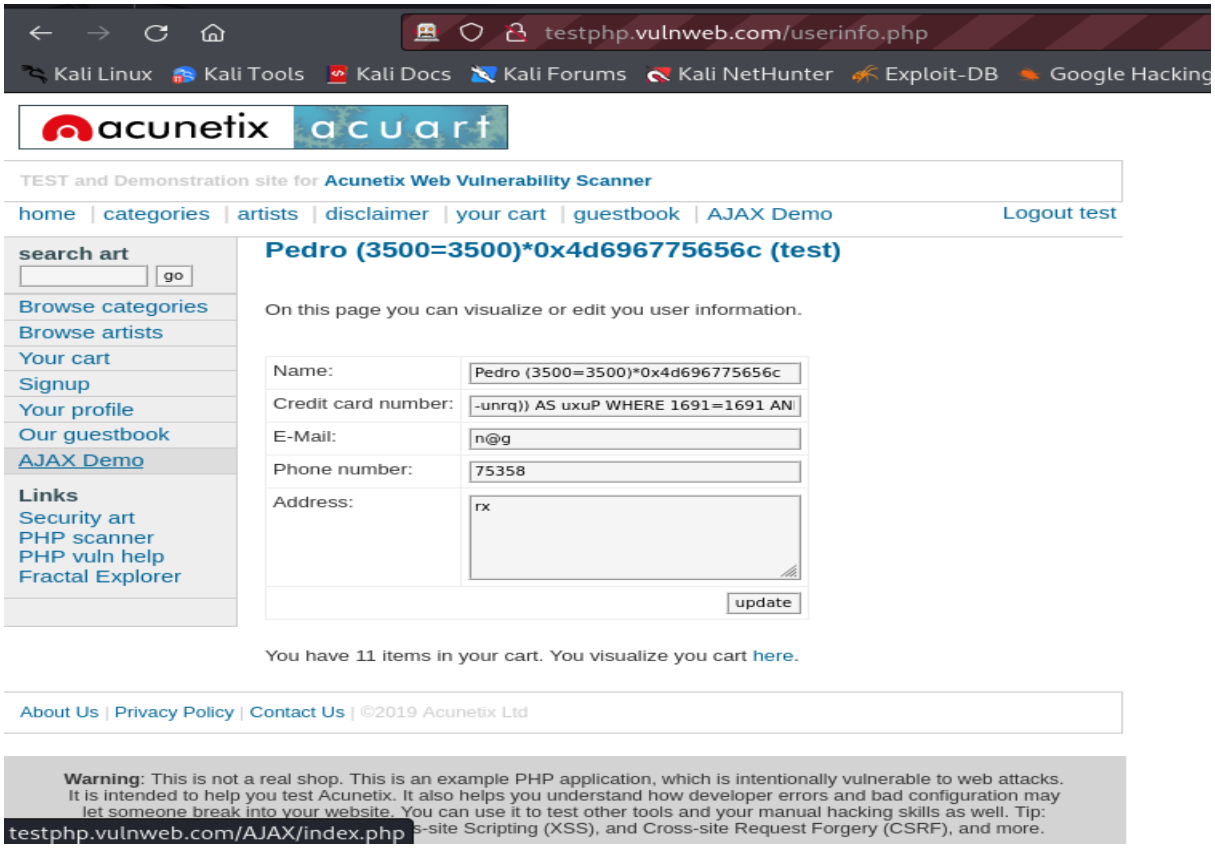
Exercise 7:

• Document any successful injections or errors encountered during fuzzing. What techniques were effective?

I attempted to inject an sql injection payload to the login page of http://testphp.vulnweb.com/login.php and the technique was effective as I was able to gain access to the page



testphp.vulnweb.com/userinfo.php



testphp.vulnweb.com/AJAX/index.php

Exercise 8:

• Prepare a report detailing the vulnerabilities discovered, your methodology, and recommendations for securing the application.

1. Introduction

This report outlines the vulnerabilities discovered during a security assessment of the application **testphp.vulnweb.com**. The purpose of the assessment was to identify potential weaknesses in the application and provide remediation suggestions to enhance its security posture. The focus was on common web vulnerabilities such as **SQL Injection (SQLi)** and **Cross-Site Scripting (XSS)**.

2. Methodology

The methodology for testing and identifying vulnerabilities in the application consisted of the following steps:

1. **Reconnaissance and Information Gathering:**
    - The application was accessed via the URL http://testphp.vulnweb.com.
    - The reconnaissance involved browsing the application, inspecting HTTP request/response data, and identifying user input fields such as login forms, search bars, and registration fields.
    - Tools used: **Burp Suite**, **OWASP ZAP** (for scanning and intercepting HTTP requests).
2. **Fuzzing Input Fields:**
    - Input fields such as **username**, **password**, and **email** were tested for vulnerabilities using **Burp Suite's Intruder** tool and **OWASP ZAP's Fuzzer**.
    - Common payloads for **SQL Injection** and **XSS** were injected into form fields to test for input validation and output encoding issues.
3. **Testing for SQL Injection (SQLi):**
    - SQL Injection was tested by submitting payloads such as ' OR '1'='1 and ' UNION SELECT null, null, version() -- into form fields that were likely to interact with a database (e.g., username, password).

- The response to these payloads was analyzed to determine if the application was vulnerable to SQL Injection.

4. **Testing for Cross-Site Scripting (XSS):**
    - Cross-Site Scripting (XSS) was tested by injecting payloads such as <script>alert('XSS')</script>, <img src="x" onerror="alert('XSS')">, and <svg/onload=alert('XSS')> into form fields and examining the behavior of the application.
    - Responses were inspected to determine if the injected payload was reflected back and executed in the browser.

3. Findings

The following vulnerabilities were identified during the testing process:

3.1. SQL Injection (SQLi) Vulnerability

- **Location**: The **signup.php** page, specifically the form fields for username and password, was found to be vulnerable to SQL Injection.
- **Issue**: The application did not properly sanitize or parameterize SQL queries, allowing user input to directly modify the structure of SQL queries.
- **Payload Used**: ' OR '1'='1
- **Impact**:
    - **Authentication Bypass**: The attacker could bypass authentication and log in without valid credentials.
    - **Data Exfiltration**: With further manipulation, an attacker could potentially extract sensitive data from the database, such as usernames, passwords, and other personal information.
    - **Data Manipulation**: Attackers could modify or delete data in the database, which could lead to data loss or corruption.

3.2. Cross-Site Scripting (XSS) Vulnerability

- **Location**: The application's **login.php** and **signup.php** pages were tested for XSS, but no reflected or stored XSS vulnerabilities were identified from the payloads used (<script>alert('XSS')</script>).

- **Issue**: While XSS payloads did not trigger any reflected XSS on the current pages tested, a proper test for **stored XSS** (e.g., submitting XSS payloads in a comment or feedback section) was not performed.
- **Impact**:
    - **User Data Exposure**: If the application reflected user input (e.g., username or comments) without proper encoding, malicious scripts could execute in the browser of other users, stealing session cookies, redirecting users, or performing other malicious actions.
    - **Session Hijacking**: Attackers could hijack user sessions by injecting malicious JavaScript that steals cookies or other session data.

3.3. Lack of Input Validation and Output Encoding

- **Location**: Both the **signup.php** and **login.php** pages were found to lack proper input validation for user inputs (e.g., username and password).
- **Issue**: User inputs were not sufficiently sanitized or validated, increasing the risk of SQL Injection, XSS, and other injection attacks.
- **Impact**: Allowing arbitrary user input without validation increases the risk of injection-based attacks (SQLi, XSS) and can allow malicious users to manipulate server-side behavior.

4. Recommendations

To mitigate the identified vulnerabilities, the following remediation steps should be taken:

4.1. Secure SQL Queries (SQL Injection Mitigation)

- **Use Prepared Statements**: All SQL queries should be prepared and parameterized to ensure user input is treated as data, not executable code. This prevents malicious users from manipulating the query structure.
- **Use ORM (Object-Relational Mapping)**: Modern frameworks often provide ORM libraries that automatically parameterize queries, reducing the risk of SQL injection.
- **Limit Database Permissions**: Ensure that the database user accounts have minimal privileges. For instance, the database account used by the web

application should not have DROP, DELETE, or other dangerous privileges unless absolutely necessary.

4.2. Prevent Cross-Site Scripting (XSS)

- **Use Output Encoding**: All user inputs that are reflected in HTML or JavaScript contexts should be properly encoded before being rendered back to the browser. This ensures that any potentially dangerous characters (e.g., <, >, &) are treated as data and not executable code.
    - **HTML Encoding**: Use a library or framework function to encode HTML entities (e.g., < becomes <, > becomes >).
    - **JavaScript Encoding**: If user input is used in JavaScript, ensure it's properly encoded or sanitized to prevent code execution.
- **Sanitize User Input**: Inputs that are used in HTML or JavaScript contexts should be sanitized to remove any dangerous elements like <script>, <img>, or javascript:.
- **Content Security Policy (CSP)**: Implement a strong **CSP** header to restrict the execution of inline JavaScript, which can mitigate certain XSS attacks.

4.3. Input Validation

- **Use Whitelisting for Input Fields**: Validate and sanitize inputs on both the client and server sides:
    - For usernames, allow only alphanumeric characters.
    - For passwords, enforce strong passwords (e.g., minimum length, at least one uppercase letter, number, and special character).
    - Ensure that no input is directly inserted into SQL queries, HTML, or JavaScript without proper validation or encoding.
- **Client-Side Validation**: Although client-side validation can improve user experience, it should never be relied upon as the only form of validation. Server-side validation is essential to ensure the integrity of the application.

4.4. Error Handling and Logging

- **Do Not Expose Detailed Error Messages**: The application should not display detailed SQL or server error messages to the user. Instead, use

generic error messages to prevent attackers from gaining insight into the application's internal workings.

- **Centralized Logging**: Implement centralized logging of error messages and input validation failures to monitor potential attack patterns.

4.5. Security Best Practices

- **Regular Security Audits**: Regularly conduct vulnerability assessments and penetration testing to identify new vulnerabilities.
- **Apply Security Patches**: Keep the web application's software and frameworks up to date with the latest security patches.
- **Session Management**: Ensure proper session management practices, including the use of secure, HttpOnly, and SameSite cookies.

5. Conclusion

The security assessment identified **SQL Injection** as a critical vulnerability on the **signup.php** page, which could lead to authentication bypass, data exfiltration, and data manipulation. Although **XSS** was not identified on the tested pages, input validation and output encoding improvements are recommended to reduce the risk of XSS attacks.

By implementing the recommended security measures, the application can significantly reduce its attack surface and enhance its overall security. Proper input validation, SQL query sanitization, and output encoding are essential to safeguarding the application against common web vulnerabilities.

Lab 9: Information Gathering with Recon-ng and Shodan

Exercise 1:

• List the modules that can be used for domain reconnaissance. What are some key modules you might consider?

recon/domains-companies/censys_companies

recon/domains-companies/pen

recon/domains-companies/whoxy_whois

recon/domains-contacts/hunter_io

recon/domains-contacts/metacrawler

| recon/domains-contacts/pen

recon/domains-contacts/pgp_search

| recon/domains-contacts/whois_pocs

| recon/domains-contacts/wikileaker

| recon/domains-domains/brute_suffix

| recon/domains-hosts/binaryedge

| recon/domains-hosts/bing_domain_api

| recon/domains-hosts/bing_domain_web

| recon/domains-hosts/brute_hosts

| recon/domains-hosts/builtwith

| recon/domains-hosts/censys_domain

| recon/domains-hosts/certificate_transparency

| recon/domains-hosts/google_site_web

| recon/domains-hosts/hackertarget

| recon/domains-hosts/mx_spf_ip

| recon/domains-hosts/netcraft

| recon/domains-hosts/shodan_hostname

| recon/domains-hosts/spyse_subdomains

| recon/domains-hosts/ssl_san

| recon/domains-hosts/threatcrowd

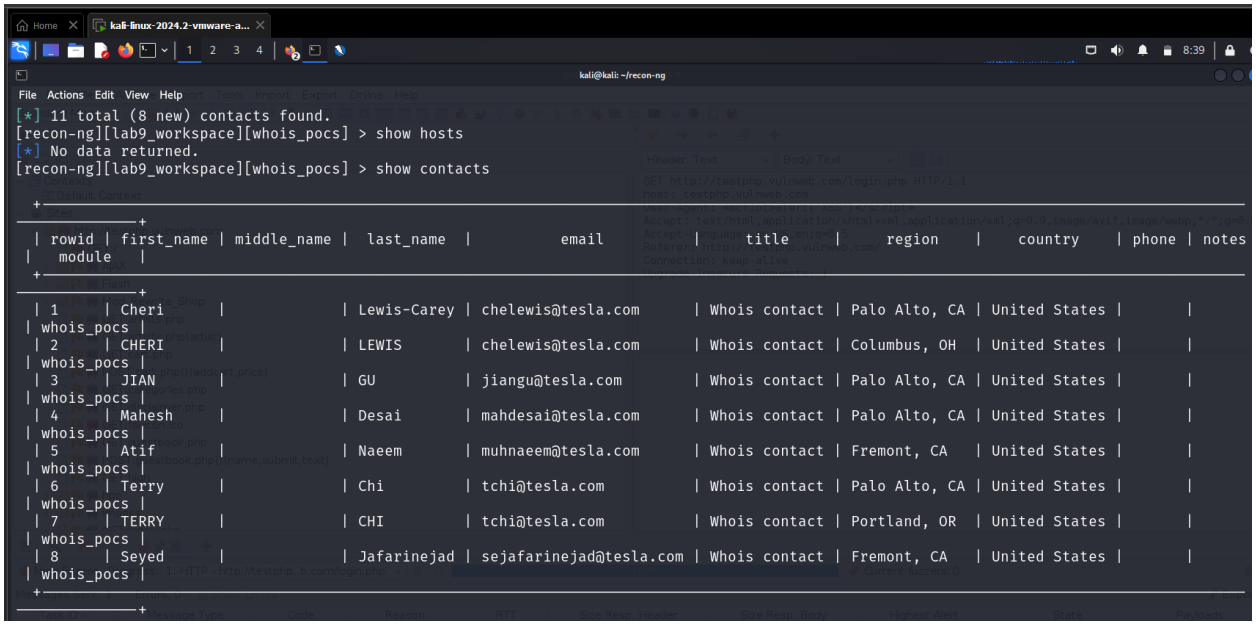
| recon/domains-hosts/threatminer

| recon/domains-vulnerabilities/ghdb

| recon/domains-vulnerabilities/xssed

Exercise 2:

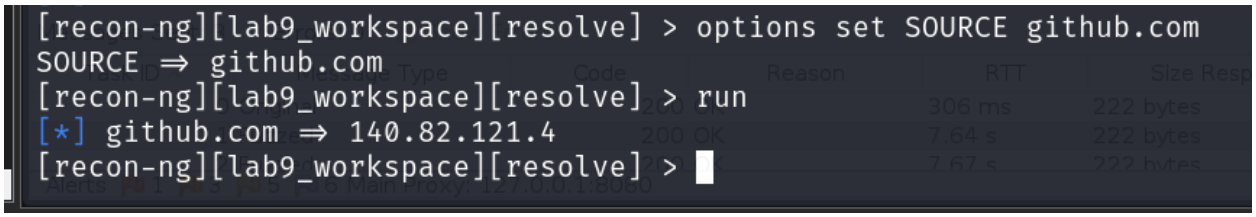
• Document the registration details obtained from the whois module. What information did you find useful?



```
[*] 11 total (8 new) contacts found.
[recon-ng][lab9_workspace][whois_pocs] > show hosts
[*] No data returned.
[recon-ng][lab9_workspace][whois_pocs] > show contacts
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
| rowid | first_name | middle_name | last_name |          email          |    title    |     region     |    country    | phone | notes |    module    |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
| 1     | Cheri      |             | Lewis-Carey | chelewis@tesla.com      | Whois contact | Palo Alto, CA  | United States |       |       | whois_pocs |
| 2     | CHERI      |             | LEWIS       | chelewis@tesla.com      | Whois contact | Columbus, OH   | United States |       |       | whois_pocs |
| 3     | JIAN       |             | GU          | jiangu@tesla.com        | Whois contact | Palo Alto, CA  | United States |       |       | whois_pocs |
| 4     | Mahesh     |             | Desai       | mahdesai@tesla.com      | Whois contact | Palo Alto, CA  | United States |       |       | whois_pocs |
| 5     | Atif       |             | Naeem       | muhnaeem@tesla.com      | Whois contact | Fremont, CA    | United States |       |       | whois_pocs |
| 6     | Terry      |             | Chi         | tchi@tesla.com          | Whois contact | Palo Alto, CA  | United States |       |       | whois_pocs |
| 7     | TERRY      |             | CHI         | tchi@tesla.com          | Whois contact | Portland, OR   | United States |       |       | whois_pocs |
| 8     | Seyed      |             | Jafarinejad | sejafarinejad@tesla.com | Whois contact | Fremont, CA    | United States |       |       | whois_pocs |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
```

Exercise 3:

• What new information was discovered about the target domain? List the subdomains or IP addresses obtained.



```
[recon-ng][lab9_workspace][resolve] > options set SOURCE github.com
SOURCE ⇒ github.com
[recon-ng][lab9_workspace][resolve] > run
[*] github.com ⇒ 140.82.121.4
[recon-ng][lab9_workspace][resolve] >
```

Exercise 4:

• Verify that your API key is working by running:

• Shodan info

Exercise 5:

• What devices were discovered related to the target domain? Provide a brief description of the findings.

I was unable to discover anything on the target because my api key was limited and cant run any search

Exercise 6:

• Perform an advanced search using two different filters. Document the results and discuss what types of devices you found.

Cant perform any search because of the restrictions