

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.

The image shows a Wireshark packet capture of an HTTP POST request. The packet list on the left shows a packet of 723 bytes on the wire (5784 bits) captured on interface eth0. The packet details pane on the right shows the following structure:

- Frame 173681: 723 bytes on wire (5784 bits), 723 bytes captured (5784 bits) on interface eth0
- Ethernet II, Src: VMware_7e:58:a5 (00:0c:29:7e:58:a5), Dst: VMware_72:12:42 (00:0c:29:72:12:42)
- Internet Protocol Version 4, Src: 192.168.29.129, Dst: 192.168.29.128
- Transmission Control Protocol, Src Port: 33854, Dst Port: 80, Seq: 1, Ack: 1, Len: 657
- Hypertext Transfer Protocol
 - HTML Form URL Encoded: application/x-www-form-urlencoded
 - Form item: "username" = "" OR 1=1"
 - Form item: "passwd" = "rrr"
 - Form item: "submit" = "Submit"

The packet bytes pane on the right shows the raw data of the packet, including the HTTP headers and the URL-encoded payload. The payload is a long string of characters, including spaces, slashes, and special characters, which is the result of the form data being encoded for transmission over the network.

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

Destination 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.

The image shows a Wireshark packet capture interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. The filter bar at the top shows 'tcp.port == 80 || udp.port == 80'. The packet list on the left shows a series of packets, with packet 1736 selected. The packet details pane on the right shows the structure of the selected packet, including Ethernet II, Internet Protocol Version 4, Transmission Control Protocol, and Hypertext Transfer Protocol. The packet bytes pane at the bottom shows the raw data of the packet, including the SQL injection payload: 'Form item: "username" = "1' OR 1=1"'. The packet list shows the following details for packet 1736:

No.	Time	Source	Destination	Protocol	Length	Info
1736	794.924370431	192.168.29.128	192.168.29.128	TCP	66	[TCP Keep-Alive ACK] Seq=2995 Ack=1145 Win=8192 Len=0 TSval=392737 TSecr=3670266877
1736	799.885327448	192.168.29.129	192.168.29.128	TCP	66	58072 → 80 [FIN, ACK] Seq=1145 Ack=2995 Win=8192 Len=0 TSval=3670281880 TSecr=392737
1736	799.885522745	192.168.29.128	192.168.29.129	TCP	66	80 → 58072 [FIN, ACK] Seq=2995 Ack=1145 Win=8192 Len=0 TSval=393957 TSecr=3670266877
1736	799.885554545	192.168.29.129	192.168.29.128	TCP	66	80 → 80 [ACK] Seq=1146 Ack=2996 Win=31872 Len=0 TSval=3670281880 TSecr=393957
1736	799.885749751	192.168.29.128	192.168.29.129	TCP	66	80 → 58072 [ACK] Seq=2996 Ack=1146 Win=8192 Len=0 TSval=393957 TSecr=3670281880
1736	834.983816314	192.168.29.129	192.168.29.128	TCP	74	33854 → 80 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=3670316978 TSecr=0 WS=128
1736	834.984667429	192.168.29.128	192.168.29.129	TCP	74	80 → 33854 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM TSval=402732 TSecr=3670316978 WS=64
1736	834.984856323	192.168.29.129	192.168.29.128	TCP	66	33854 → 80 [ACK] Seq=1 Ack=1 Win=32120 Len=0 TSval=3670316979 TSecr=402732
1736	834.985399171	192.168.29.129	192.168.29.128	HTTP	723	POST /owaspbricks/login-1/index.php HTTP/1.1 (application/x-www-form-urlencoded)
1736	834.98592550	192.168.29.128	192.168.29.129	TCP	66	80 → 33854 [ACK] Seq=1 Ack=658 Win=7168 Len=0 TSval=402732 TSecr=3670316980
1736	834.9859393476	192.168.29.128	192.168.29.129	HTTP	1602	HTTP/1.1 200 OK (text/html)
1736	834.985959553	192.168.29.129	192.168.29.128	TCP	66	33854 → 80 [ACK] Seq=658 Ack=1537 Win=31872 Len=0 TSval=3670316984 TSecr=402733
1736	845.099452770	192.168.29.129	192.168.29.128	TCP	66	[TCP Keep-Alive ACK] Seq=657 Ack=1537 Win=31872 Len=0 TSval=3670327174 TSecr=402733
1736	845.100259297	192.168.29.128	192.168.29.129	TCP	66	[TCP Keep-Alive ACK] Seq=1537 Ack=658 Win=7168 Len=0 TSval=405281 TSecr=3670316984
1736	849.10819608	192.168.29.129	192.168.29.128	TCP	66	33854 → 80 [FIN, ACK] Seq=658 Ack=1537 Win=31872 Len=0 TSval=3670331985 TSecr=405281
1736	849.911759996	192.168.29.128	192.168.29.129	TCP	66	80 → 33854 [FIN, ACK] Seq=1537 Ack=659 Win=7168 Len=0 TSval=406484 TSecr=3670331985
1736	849.911883447	192.168.29.129	192.168.29.128	TCP	66	33854 → 80 [ACK] Seq=659 Ack=1538 Win=31872 Len=0 TSval=3670331986 TSecr=406484
1737	934.376127549	192.168.29.129	192.168.29.128	TCP	74	52116 → 80 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=3670416451 TSecr=0 WS=128

The packet details pane for packet 1736 shows the following structure:

- Ethernet II, Src: VMware_7e:58:a5 (08:0c:29:7e:58:a5), Dst: VMware_72:12:42 (08:0c:29:72:12:42)
- Internet Protocol Version 4, Src: 192.168.29.129, Dst: 192.168.29.128
- Transmission Control Protocol, Src Port: 33854, Dst Port: 80, Seq: 1, Ack: 1, Len: 657
- Hypertext Transfer Protocol
- HTML Form URL Encoded, application/x-www-form-urlencoded
- Form item: "username" = "1' OR 1=1"
- Form item: "passwd" = "rrrr"
- Form item: "submit" = "Submit"

The packet bytes pane shows the raw data of the packet, including the SQL injection payload: 'Form item: "username" = "1' OR 1=1"'. The packet list shows the following details for packet 1736:

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.

File
Edit
View
Go
Capture
Analyze
Statistics
Telephony
Wireless
Tools
Help

udp.port

No.	Time	Source	Destination	Protocol	Length	Info
1728	748.683782262	192.168.29.1	192.168.29.129	ICMP	125	Destination unreachable (Port unreachable)
1728	748.684623244	192.168.29.129	192.168.29.1	DNS	85	Standard query 0x2204 AAA push.services.mozilla.com
1728	748.684633569	192.168.29.129	192.168.29.1	DNS	85	Standard query 0x8ba1 AAAA push.services.mozilla.com
1728	748.684831841	192.168.29.1	192.168.29.129	ICMP	113	Destination unreachable (Port unreachable)
1728	748.684832142	192.168.29.1	192.168.29.129	ICMP	113	Destination unreachable (Port unreachable)
1728	748.684834376	192.168.29.129	192.168.29.1	DNS	85	Standard query 0x2204 AAA push.services.mozilla.com
1728	748.685059188	192.168.29.129	192.168.29.1	DNS	85	Standard query 0x8ba1 AAAA push.services.mozilla.com
1728	748.685293458	192.168.29.1	192.168.29.129	ICMP	113	Destination unreachable (Port unreachable)
1728	748.685293708	192.168.29.1	192.168.29.129	ICMP	113	Destination unreachable (Port unreachable)
1728	748.686195550	192.168.29.129	192.168.29.1	DNS	97	Standard query 0x6b09 AAA push.services.mozilla.com.localdomain
1728	748.688170376	192.168.29.1	192.168.29.129	ICMP	125	Destination unreachable (Port unreachable)
1728	748.689683786	192.168.29.129	192.168.29.1	DNS	97	Standard query 0x37b7 AAAA push.services.mozilla.com.localdomain
1728	748.691431366	192.168.29.1	192.168.29.129	ICMP	125	Destination unreachable (Port unreachable)
1728	748.691599045	192.168.29.129	192.168.29.1	DNS	97	Standard query 0x6b09 AAA push.services.mozilla.com.localdomain
1728	748.693709785	192.168.29.129	192.168.29.1	DNS	97	Standard query 0x37b7 AAAA push.services.mozilla.com.localdomain
1728	748.693986759	192.168.29.1	192.168.29.129	ICMP	125	Destination unreachable (Port unreachable)
1728	748.693987019	192.168.29.1	192.168.29.129	ICMP	125	Destination unreachable (Port unreachable)
1728	748.702597910	192.168.29.129	192.168.29.1	DNS	85	Standard query 0x3f07 AAA push.services.mozilla.com
1728	748.703129897	192.168.29.129	192.168.29.1	DNS	85	Standard query 0x6783 AAAA push.services.mozilla.com

Frame 173673: 125 bytes on wire (1000 bits), 125 bytes captured (1000 bits) on interface eth0

Ethernet II, Src: VMware_C8:00:0E:11 (08:00:56:c8:00:0e), Dst: VMware_7e:58:a5 (08:0c:29:7e:58:a5)

Internet Protocol Version 4, Src: 192.168.29.1, Dst: 192.168.29.129

Internet Control Message Protocol

Domain Name System (query)

Transaction ID: 0xb0b54

Flags: 0x010 Standard query

Questions: 1

Answer RRs: 0

Authority RRs: 0

Additional RRs: 0

Queries

push.services.mozilla.com.localdomain: type AAAA, class IN

Text item (text), 43 bytes

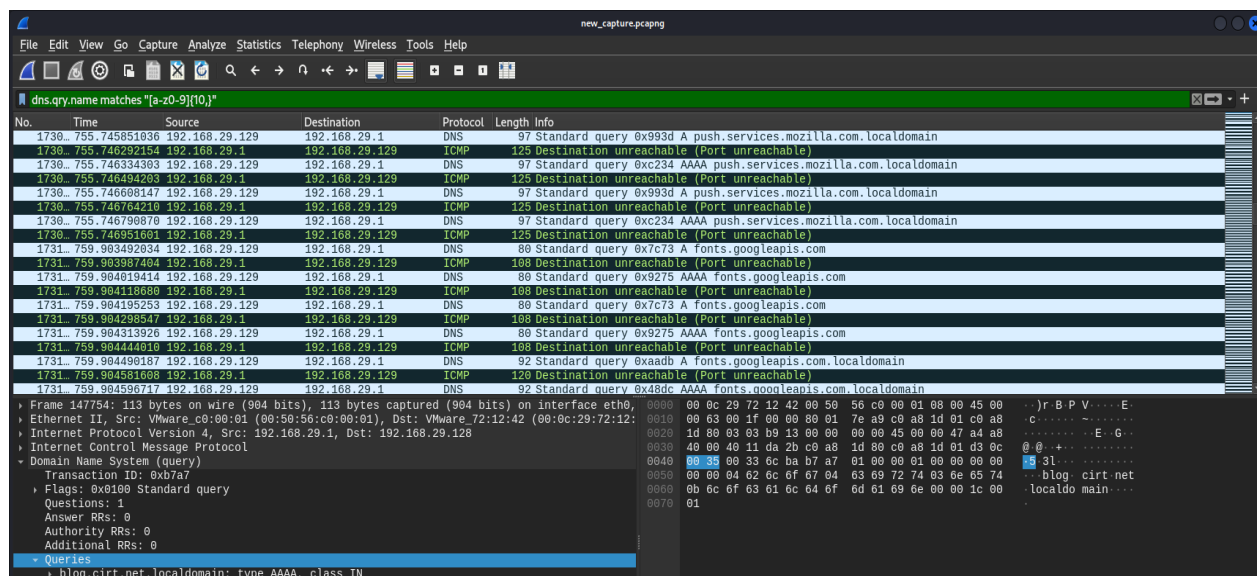
Packets: 1558338 - Displayed: 1084 (0.1%)

Profile: Default

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.



The image shows a Wireshark packet capture window titled "new_capture.pcapng". The filter bar is set to "dns.qry.name matches '[a-z0-9]{10,}'". The packet list shows a series of DNS queries and ICMP responses. The packet details pane is expanded for packet 1731, showing a DNS Standard query for "blog.cirt.net.localdomain". The packet bytes pane shows the raw data of the DNS query.

No.	Time	Source	Destination	Protocol	Length	Info
1730	755.745851836	192.168.29.129	192.168.29.1	DNS	97	Standard query 0x993d A push.services.mozilla.com.localdomain
1730	755.746292154	192.168.29.1	192.168.29.129	ICMP	125	Destination unreachable (Port unreachable)
1730	755.746334393	192.168.29.129	192.168.29.1	DNS	97	Standard query 0xc234 AAAA push.services.mozilla.com.localdomain
1730	755.746494203	192.168.29.1	192.168.29.129	ICMP	125	Destination unreachable (Port unreachable)
1730	755.746608147	192.168.29.129	192.168.29.1	DNS	97	Standard query 0x993d A push.services.mozilla.com.localdomain
1730	755.746764210	192.168.29.1	192.168.29.129	ICMP	125	Destination unreachable (Port unreachable)
1730	755.746790870	192.168.29.129	192.168.29.1	DNS	97	Standard query 0xc234 AAAA push.services.mozilla.com.localdomain
1730	755.746951601	192.168.29.1	192.168.29.129	ICMP	125	Destination unreachable (Port unreachable)
1731	759.903492834	192.168.29.129	192.168.29.1	DNS	80	Standard query 0x7c73 A fonts.googleapis.com
1731	759.903987404	192.168.29.1	192.168.29.129	ICMP	108	Destination unreachable (Port unreachable)
1731	759.904019414	192.168.29.129	192.168.29.1	DNS	80	Standard query 0x9275 AAAA fonts.googleapis.com
1731	759.904116080	192.168.29.1	192.168.29.129	ICMP	108	Destination unreachable (Port unreachable)
1731	759.904199253	192.168.29.129	192.168.29.1	DNS	80	Standard query 0x7c73 A fonts.googleapis.com
1731	759.904290547	192.168.29.1	192.168.29.129	ICMP	108	Destination unreachable (Port unreachable)
1731	759.904313926	192.168.29.129	192.168.29.1	DNS	80	Standard query 0x9275 AAAA fonts.googleapis.com
1731	759.904444010	192.168.29.1	192.168.29.129	ICMP	108	Destination unreachable (Port unreachable)
1731	759.904490187	192.168.29.129	192.168.29.1	DNS	92	Standard query 0xaadb A fonts.googleapis.com.localdomain
1731	759.904581608	192.168.29.1	192.168.29.129	ICMP	120	Destination unreachable (Port unreachable)
1731	759.904596717	192.168.29.129	192.168.29.1	DNS	92	Standard query 0x48dc AAAA fonts.googleapis.com.localdomain

Frame 147754: 113 bytes on wire (904 bits), 113 bytes captured (904 bits) on interface eth0
Ethernet II, Src: VMware_c0:00:01 (00:50:56:c0:00:01), Dst: VMware_72:12:42 (00:0c:29:72:12:42)
Internet Protocol Version 4, Src: 192.168.29.1, Dst: 192.168.29.128
Internet Control Message Protocol
Domain Name System (query)
Transaction ID: 0xb7a7
Flags: 0x100 Standard query
Questions: 1
Answer RRs: 0
Authority RRs: 0
Additional RRs: 0
Queries
blog.cirt.net.localdomain: type AAAA, class IN

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.

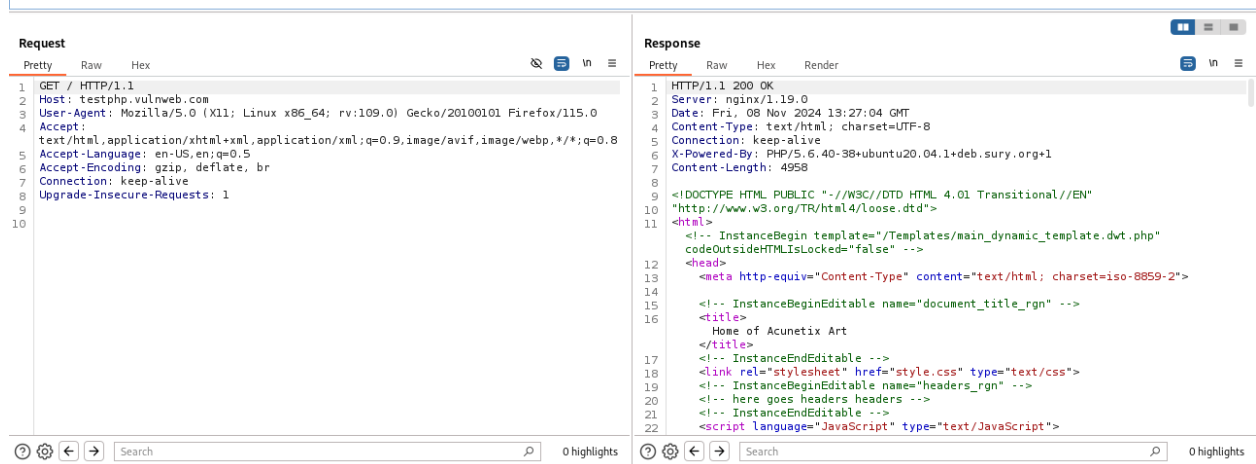
Conclusion:

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>

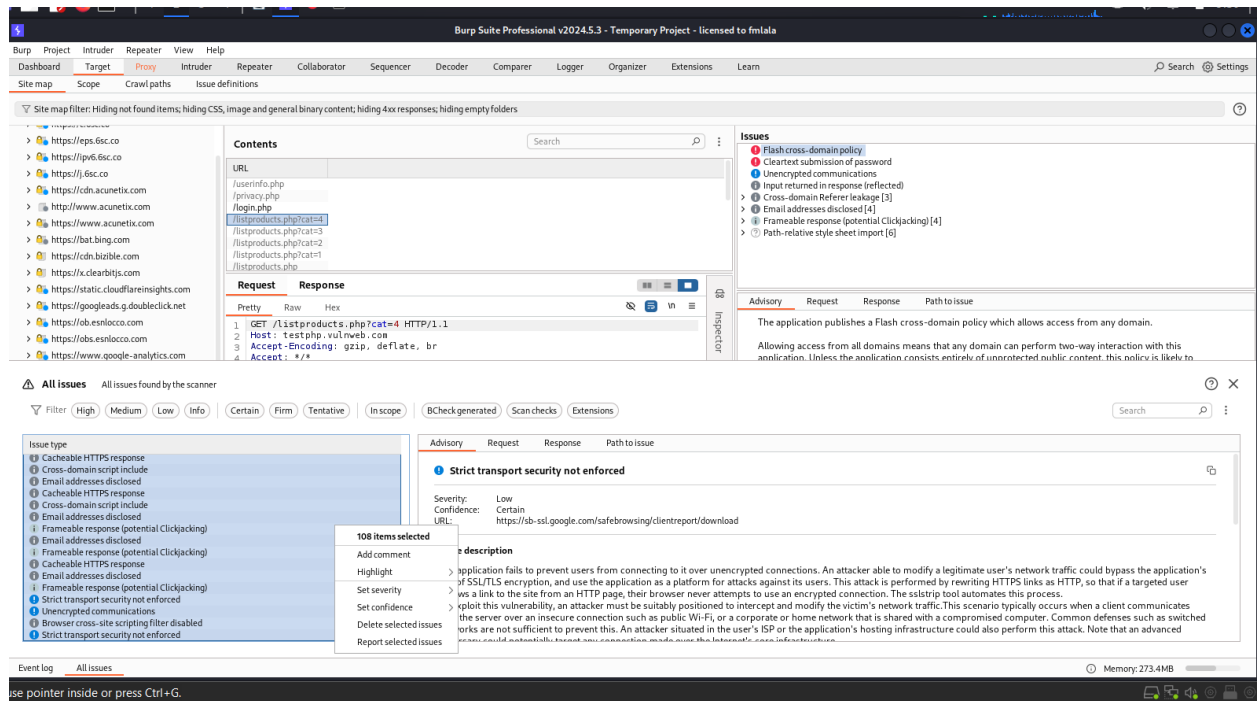
<http://testphp.vulnweb.com/listproducts.php?cat=4>

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 Password:

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

Advisory	Request	Response	Path to issue
<div> <div> <div> <div></div> <div>Cleartext submission of password</div> </div> <div></div> </div> </div>			
Severity:	High		
Confidence:	Certain		
URL:	http://testphp.vulnweb.com/login.php		
Issue detail			
The page contains a form with the following action URL, which is submitted over clear-text HTTP:			
<ul style="list-style-type: none"> http://testphp.vulnweb.com/userinfo.php 			
The form contains the following password field:			
<ul style="list-style-type: none"> pass 			

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):

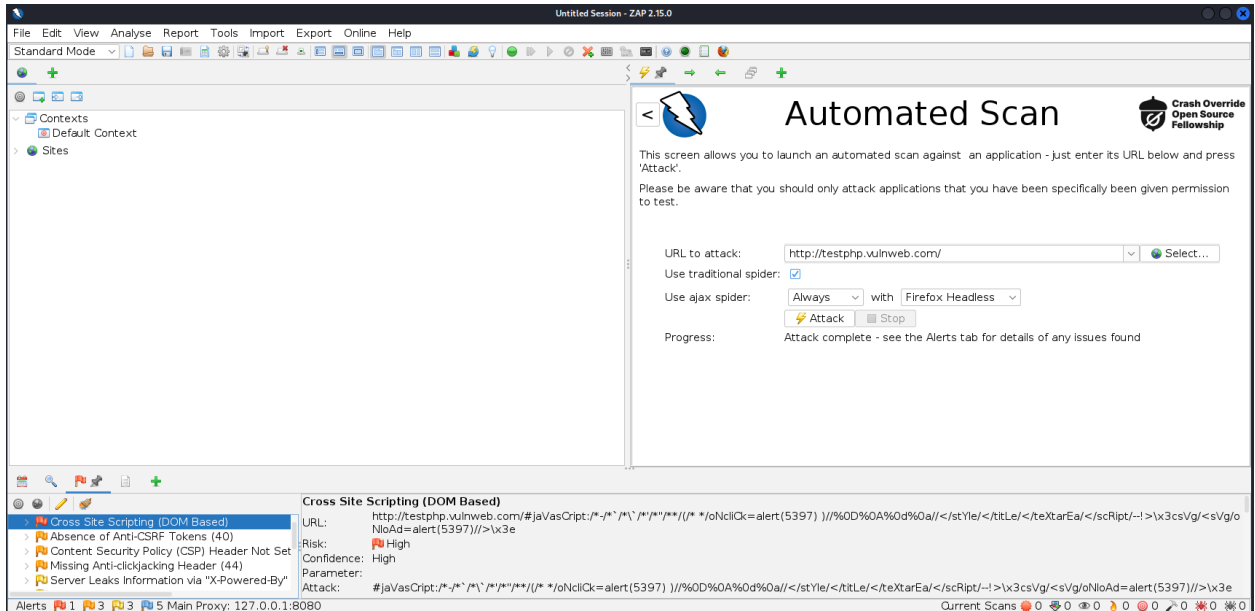
- **Impact:** XSS can lead to the execution of arbitrary scripts in a user's browser, which can be used to steal session cookies, redirect the user to malicious sites, or perform actions on behalf of the user without their consent.

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



acunetix

acuart

TEST and Demonstration site for [Acunetix Web Vulnerability Scanner](#)

[home](#) | [categories](#) | [artists](#) | [disclaimer](#) | [your cart](#) | [guestbook](#) | [AJAX Demo](#)

search art

go

Browse categories

Browse artists

Your cart

Signup

[Your profile](#)

[Our guestbook](#)

[AJAX Demo](#)

Links

[Security art](#)

[PHP scanner](#)

[PHP vuln help](#)

[Fractal Explorer](#)

If you are already registered please enter your login information below:

Username :

Password :

You can also

Signup disabled

This connection is not secure.
Logins entered here could be compromised. [Learn More](#)

the password **test**.

[About Us](#) | [Privacy Policy](#) | [Contact Us](#) | ©2019 Acunetix Ltd

Warning: This is not a real shop. This is an example PHP application, which is intentionally vulnerable to web attacks. It is intended to help you test Acunetix. It also helps you understand how developer errors and bad configuration may let someone break into your website. You can use it to test other tools and your manual hacking skills as well. Tip: Look for potential SQL Injections, Cross-site Scripting (XSS), and Cross-site Request Forgery (CSRF), and more.

testphp.vulnweb.com/userinfo.php



acunetix

acuart

TEST and Demonstration site for [Acunetix Web Vulnerability Scanner](#)

[home](#) | [categories](#) | [artists](#) | [disclaimer](#) | [your cart](#) | [guestbook](#) | [AJAX Demo](#) [Logout test](#)

search art

go

Browse categories

Browse artists

Your cart

Signup

[Your profile](#)

[Our guestbook](#)

[AJAX Demo](#)

Links

[Security art](#)

[PHP scanner](#)

[PHP vuln help](#)

[Fractal Explorer](#)

Pedro (3500=3500)*0x4d696775656c (test)

On this page you can visualize or edit you user information.

Name:

Credit card number:

E-Mail:

Phone number:

Address:

update

You have 11 items in your cart. You visualize you cart [here](#).

[About Us](#) | [Privacy Policy](#) | [Contact Us](#) | ©2019 Acunetix Ltd

Warning: This is not a real shop. This is an example PHP application, which is intentionally vulnerable to web attacks. It is intended to help you test Acunetix. It also helps you understand how developer errors and bad configuration may let someone break into your website. You can use it to test other tools and your manual hacking skills as well. Tip: Look for potential SQL Injections, Cross-site Scripting (XSS), and Cross-site Request Forgery (CSRF), and more.

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>`, ``, 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>`, ``, 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

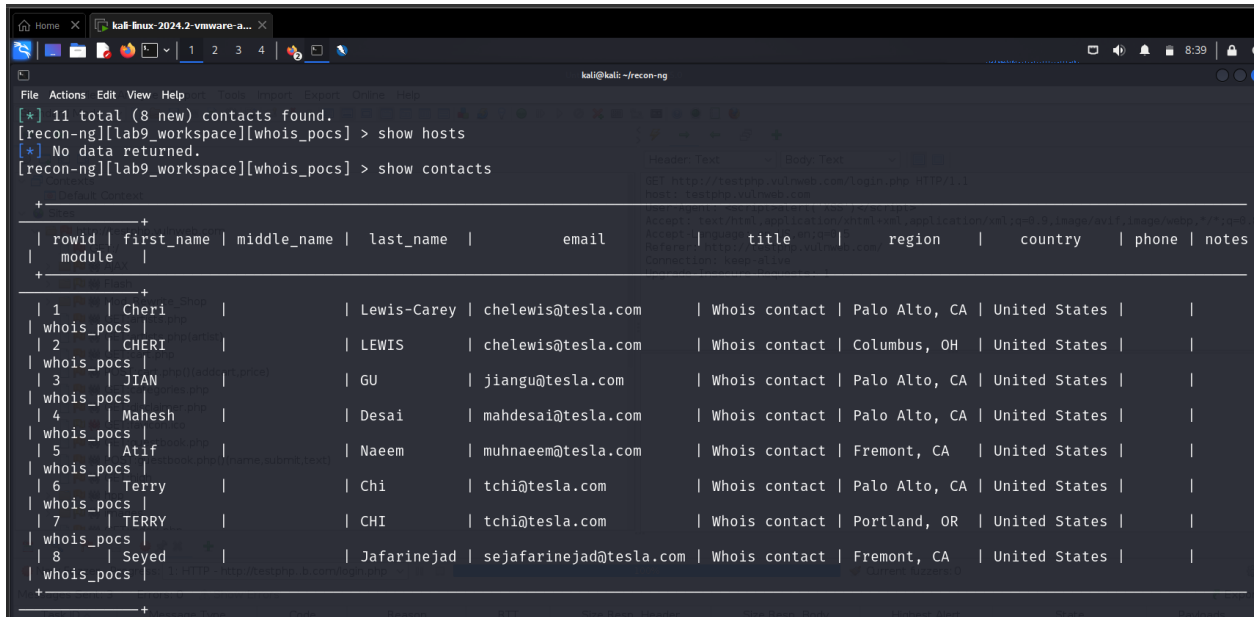
| recon/domains-hosts/threatminer

| 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?

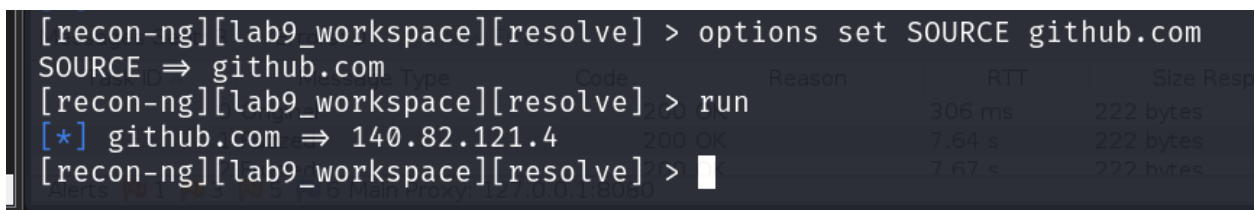


```
[*] 11 total (8 new) contacts found.
[recon-ng][lab9_workspace][whois_pocs] > show hosts
[*] No data returned.
[recon-ng][lab9_workspace][whois_pocs] > show contacts
```

rowid	first_name	middle_name	last_name	email	title	region	country	phone	notes
1	Cheri		Lewis-Carey	chelewis@tesla.com	Whois contact	Palo Alto, CA	United States		
2	CHERI		LEWIS	chelewis@tesla.com	Whois contact	Columbus, OH	United States		
3	JIAN		GU	jiangu@tesla.com	Whois contact	Palo Alto, CA	United States		
4	Mahesh		Desai	mahdesai@tesla.com	Whois contact	Palo Alto, CA	United States		
5	Atif		Naeem	muhnaeem@tesla.com	Whois contact	Fremont, CA	United States		
6	Terry		Chi	tchi@tesla.com	Whois contact	Palo Alto, CA	United States		
7	TERRY		CHI	tchi@tesla.com	Whois contact	Portland, OR	United States		
8	Seyed		Jafarinejad	sejafarinejad@tesla.com	Whois contact	Fremont, CA	United States		

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
```

Exercise 4:

- Verify that your API key is working by running:
- Shodan info

```
(kali㉿kali)-[~]  
$ shodan init AUhZkipQ9Sha7uvEySSK6lr8cAnPutAe  
Successfully initialized  
  
(kali㉿kali)-[~]  
$ shodan info  
Query credits available: 0  
Scan credits available: 0
```

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