Projet Stegobox - PentaCrevits

CASTANEDA Hugo
CREVITS François
JARRETIER-YUSTE Adrien

Introduction

L'objectif de ce tuto est de créer une clé bootable délivrant une application de stéganographie (cacher des messages dans une image) à une adresse donnée. Un serveur web doit donc être mis en place et permettre à l'utilisateur de réaliser les actions suivantes:

- uploader et sélectionner une image
- saisir un texte à cacher dans l'image sélectionnée
- cacher le texte et télécharger l'image le contenant

Par ailleurs, l'adresse IP à laquelle l'application est disponible doit être affichée automatiquement au boot de la clé. Le choix des différentes technologies utilisées est également pensé afin de produire un système léger.

Tutoriel

Nous allons travailler depuis un système Linux, par exemple Ubuntu.

Nous installerons un système linux complet mais minimaliste sur une clé usb. Lors de l'opération la clé sera entièrement formatée, il faut donc s'assurer de n'avoir aucune donnée à récupérer.

Les commandes à exécuter seront indiquée comme ceci : `commande`

Clé bootable

- La première chose à faire est de passer root `su` ou `sudo -i`
- Avec la clé branchée, nous allons créer un nouveau système de fichier dessus, assurez vous de l'emplacement de votre clé (pour nous c'est /dev/sdb1) cela peut changer si vous avez d'autres périphériques de stockage sur la machine.
- Lancez l'utilitaire de partitionnement `fdisk /dev/sdb`
- Une première partition en ext4 `mkfs.ext4 /dev/sedb1`
- Quittez l'utilitaire en sauvegardant `w`
- Créez un point de montage `mkdir fs`
- Montez la partition nouvellement créée `mount /dev/sdb1 fs`
- Installez debootstrap : `apt update && apt install debootstrap`

- Lancez debootstrap pour installer debian Jessie sur la partition `debootstrap --arch amd64 jessie fs http://ftp.fr.debian.org/debian`
- Liez le proc et le dev de votre poste a ceux de la clé usb

```
mount -t proc none fs/proc
mount -o bind /dev fs/dev
```

- Nous allons maintenant utiliser ce nouveau système comme répertoir racine afin de le configurer, changez le répertoire racine `chroot fs`
- Initialisez un mot de passe au compte root `passwd`
- Installez un noyau `apt update && apt install linux-image-amd64`
- Recuperez l'UUID de la clé USB : `blkid`
- Editez **/etc/fstab** `nano /etc/fstab`

```
proc /proc proc defaults
UUID=xxxxxxxxxxx    /    ext4 errors=remount-ro 0 1
```

- Initialisez le hostname `echo "mykeykey" > /etc/hostname`
- Editez le fichier **/etc/network/interfaces**

```
auto lo
iface lo inet loopback

allow-hotplug eth0
auto eth0
iface eth0 inet dhcp

allow-hotplug eth1
auto eth1
iface eth1 inet dhcp

allow-hotplug eth2
auto eth2
iface eth2 inet dhcp
```

Nous avons édité le fichier pour les interfaces 0 à 2 mais il se peut que l'interface réseau de la machine sur laquelle vous brancherez la clé ne soit pas l'une de ces 3 il faudra alors éditer à nouveau ce fichier.

- Supportez un clavier azerty `apt install console-data`
- (toujours chroote) Installez grub2 `apt install grub2` (sur /dev/sdb)
- Quittez le chroot `exit`
- Démontez les trois montages `umount fs/{dev,proc} && umount fs`

Afficher l’IP du serveur de l’application

Nous allons au démarrage du système, sur l’écran de login, afficher l’adresse ip sur laquelle écoute le serveur.

Le texte écrit sur l'écran de login est contenu dans le fichier **/etc/issue,** c’est dans celui-ci qu’il va donc falloir écrire l’adresse ip.

Pour obtenir automatiquement l’adresse ip de notre machine sur le réseau nous utiliserons le script **etc/rc.local** qui est exécuté par le système à la fin du démarrage.

Modifiez **etc/rc.local** en ajoutant ce contenu :

```
IP=$(/sbin/ifconfig eth0 | grep 'inet addr:' | cut -d: -f2 | awk '{
print $1}')
echo "eth0 IP: $IP" > /etc/issue

IP=$(/sbin/ifconfig eth1 | grep 'inet addr:' | cut -d: -f2 | awk '{
print $1}')
echo "eth1 IP: $IP" >> /etc/issue

IP=$(/sbin/ifconfig eth2 | grep 'inet addr:' | cut -d: -f2 | awk '{
print $1}')
echo "eth2 IP: $IP" >> /etc/issue


exit 0
```

Côté client

Notre partie client utilisera les technologies standards front-end de déclaration HTML/CSS et de traitement via Javascript. Comme il s'agit d'une partie très modulable, nous nous contenterons d'expliquer le fonctionnement des différentes interactions telles qu'elles ont été implémentées.

Contenu de la page

La page est composée de deux "panels" de base.

- Sélection de l'image

  Ce panel contient un espace de sélection d'image. Toutes les images du serveur y sont affichées. Il suffit de cliquer sur l'une d'elles pour indiquer à l'application que l'on souhaite l'utiliser.

  Un bouton "Upload" précédé d'un input de sélection de fichier permet d'uploader des fichiers dans le serveur. Lorsqu'une image est uploadée, l'espace de sélection est mis à jour avec la nouvelle image.

- Texte à cacher

  Ce panel contient 3 éléments. Un "textarea" permet d'inscrire le texte à cacher. En appuyant sur le bouton "Hide" les informations à combiner sont envoyés au serveur qui répond avec l'url de l'image générée. Cette URL est ensuite accessible via un lien généré sous le bouton.

La page contient un descripteur de style CSS permettant d'organiser les différents "panel" et de faire ressortir les éléments importants (boutons, image sélectionnée, etc.).

Traitement des entrées utilisateur

- Sélection d'une image

  Lorsque l'utilisateur sélectionne une image, une variable globale est mise à jour pour indiquer au reste de l'application (traitements futurs) l'image à utiliser.

- Upload d'une image

  Lorsque l'utilisateur appui sur le bouton "Upload", l'application récupère les informations relatives à l'image à enregistrer stockée dans la propriété file du composant input de fichier. Ces informations sont insérées dans un formdata et envoyées au serveur. Ce processus que nous avons "javascriptifié" est un processus automatique lorsqu'il est utilisé dans un formulaire html. Ce choix de conception est dû à notre volontée de conserver une expérience utilisateur raisonnable en minimisant le nombre de rechargement de la page.

- Cacher le texte

  Lorsque l'utilisateur souhaite cacher son texte (en appuyant sur le bouton "Hide"), l'application envoie le texte ainsi que l'url de l'image sélectionnée (enregistrée dans sa variable globale) au serveur qui répond avec l'url de l'image sélectionnée (comme décrit précédemment).

Le traitement des clics utilisateurs est initié par la propriété “onClic” dans le cas des boutons ou par la surcharge de l'événement “click” via jQuery lorsqu'il s'agit d'éléments plus basiques du DOM (exemple: les images à sélectionner).

La communication avec le serveur se fait via des requêtes Ajax par le biais de jQuery.

Serveur Web et CGI

Nous allons utiliser le serveur web Lighttpd, qui se veut léger et rapide, et activer les CGIs afin de pouvoir exécuter des scripts bash en accédant à certaines adresses.

- Installez Lighttpd `apt install lighttpd`
- Vous pouvez vérifier que l'installation s'est déroulée correctement en lançant lighttpd puis en vous rendant sur le localhost `sudo service lighttpd start`
- Activez les CGIs `sudo lighty-enable-mod cgi`
  les scripts devront être placés dans **cgi/bin**
- Modifiez le fichier de configuration **etc/lighttpd/conf-enabled/10-cgi.conf** pour traiter les requêtes vers **cgi-bin/<fichier.sh>**

```
server.modules += ( "mod_cgi" )

$HTTP["url"] =~ "^/cgi-bin/" {
        cgi.assign = ( ".sh" => "" )
}

## Warning this represents a security risk, as it allow to execute any
file
## with a .pl/.py even outside of /usr/lib/cgi-bin.
#
#cgi.assign      = (
#       ".pl"  => "/usr/bin/perl",
#       ".py"  => "/usr/bin/python",
#)
```

Notez que l'on pourrait utiliser du perl ou du python (ou autre chose) en utilisant les lignes commentées à la fin du fichier.

- Rechargez la configuration lighttpd `sudo service lighttpd force-reload`

Scripts Bash

L'application se base sur trois scripts afin de réaliser la stéganographie.

1. uploadImage.sh

Permet d'ajouter une image (non cryptée) sur le site. Traite une requête POST contenant un formulaire multipart. Parser la requête est donc délicat puisqu'il y a plusieurs lignes ne contenant pas d'informations utiles (pour nous) à enlever avant de pouvoir enregistrer l'image.

```
#!/bin/bash

if [ "$REQUEST_METHOD" = "POST" ]; then
  TMPOUT=tmp/tmpImgUpload
  cat >$TMPOUT

  # Get the line count
  LINES=$(wc -l $TMPOUT | cut -d ' ' -f 1)

  filename=$(head -2 $TMPOUT | tail -1 | sed 's/.*filename="\(.*\)".*/\1/g')

  # Remove the first four lines
  tail -$((LINES - 4)) $TMPOUT >$TMPOUT.1

  # Remove the last line
  head -$((LINES - 5)) $TMPOUT.1 >$TMPOUT

  # Copy everything but the new last line to a temporary file
  head -$((LINES - 6)) $TMPOUT > ../imgs/$filename

  # Copy the new last line but remove trailing \r\n
  tail -1 $TMPOUT | perl -p -i -e 's/\r\n$//' >> ../imgs/$filename

  echo 'Content-type: application/json'
  echo ''

  echo "{\"filePath\" : \"imgs/$filename\"}"

fi

exit 0
```

2. whatImages.sh

Traite une requête GET effectuée au chargement de la page. Renvoie le tableau des images déjà uploadées en JSON.

```
#!/bin/bash

echo 'Content-type: application/json'
echo ''

echo '['

images=""

for image in ../imgs/*
do

  images=$images'"imgs/'$(basename $image)'"'

done

images=$(echo $images | sed 's/""/","/g')

echo $images

echo ']'

exit 0
```

3. hideImage.sh

Traite une requête POST contenant le texte à cacher ainsi que l'adresse de l'image dans laquelle le cacher. Nécessite de reconvertir en UTF-8 ce qui a été url encodé lors de la requête à l'aide d'une regex.
L'appel à steghide est ensuite effectué avec les informations obtenues. Nous répondons avec le bon en-tête en cas d'erreur et le lien vers l'image modifiée. Le nom de cette image est déterminé aléatoirement et stockée dans /messagesCaches.

```
#!/bin/bash

decodeURL() { printf "%b\n" "$(sed 's/+/ /g;
s/%\([1-9a-f][0-9a-f]\)/\\x\1/gi;')"; }

if [ "$REQUEST_METHOD" = "POST" ]; then

  TMPOUT=tmp/tmpHideImage
  OUPUT_FILE=$(mktemp -up messagesCaches XXXXXXXXXX)

  cat >$TMPOUT
  echo '' >> $TMPOUT

  text=$(decodeURL <<< $(sed 's/.*text=\([^&]*\).*/\1/g' $TMPOUT))
  image=$(decodeURL <<< $(sed 's/.*image=\([^&]*\).*/\1/g' $TMPOUT))

  OUPUT_FILE=$OUPUT_FILE.$(echo $image | sed 's/.*\.\(.*\)/\1/')

  echo $text | sed 's/%0A/\n/g' > $TMPOUT.text

  steghide embed -ef $TMPOUT.text -cf ../$image -sf ../$OUPUT_FILE -p 'password'
-f -q

  if (( $? == 1 ))
  then

       echo 'status: 500'
       echo ''

  else

       echo 'Content-type: application/json'
       echo ''

       echo "{\"filePath\" : \"$OUPUT_FILE\"}"

  fi

fi

exit $?
```

Conclusion

Ce tutoriel permet bien d'atteindre les objectifs fixés en introduction. Néanmoins, certains points se sont avérés assez problématiques.
Pour afficher l'IP de l'application, on se contente d'afficher un nombre prédéterminé d'interfaces, sans aucune forme de détection, alors que littéralement n'importe laquelle peut être utilisée.
Parser les requêtes POST au sein de scripts bash sans avoir recours à Python ou à PHP afin de garder un système léger a également été un exercice complexe.

Plusieurs améliorations sont envisageables pour l'application. Un système d'exploitation encore plus minimaliste tel que *Gentoo* pourrait être installé, bien que nous n'y soyons pas parvenu après quelques essais infructueux. Cela serait une première étape pour réduire l'empreinte mémoire du système.
Il est également possible d'ajouter des fonctionnalités pour l'utilisateur, comme récupérer le texte caché dans une image en l'uploadant. On peut alors aussi permettre à l'utilisateur de renseigner le mot de passe requis par steghide plutôt que d'en utiliser un par défaut.

Taille totale du système avec toutes les applications : 763 Mo

Application web : 132 Ko

Récapitulatif des applications installées :

- Lighttpd
- Steghide
- Jquery (ainsi l'application peut fonctionner sur un réseau local sans internet)
- SSH
- caca-utils pour visionner les images dans le terminal avec cacaview