MEUNIER Joran-Marc
BADAY Adham
SLIMANI Hayet

M1 Projet Réseau 2020

Etude de Websnarf

1. Websnarf est un programme simulant un serveur web acceptant toutes les connexions, il écoute par défaut sur le port 80 et sauvegarde chaque tentative de connexion avec la date, l'heure, l'adresse locale et distante ainsi que la requête, il ferme ensuite la connexion puis recommence à écouter sur le port. Le programme a d'abord été créé pour monitorer la propagation de l'infection de serveur par le ver Code Red.

2. *Exemple d'utilisation :*

On lance d'abord websnarf et lorsque l'on essaye de se connecter via un navigateur par exemple, le programme affiche les informations de connexion (date, heure, adresses, requête)

Lorsqu'une connexion est effectuée, websnarf attend 5 secondes si aucune requête n'est reçue avant de fermer la connexion et de relancer l'écoute.

On voit ici que la connexion s'est fait à partir de la machine locale vers la machine locale.

```
websnarf v1.04 listening on port 80 (timeout=5 secs)
12/31 23:45:51 127.0.0.1       -> 127.0.0.1      : GET / HTTP/1.1

12/31 23:45:51 127.0.0.1       -> 127.0.0.1      : GET / HTTP/1.1

12/31 23:45:51 127.0.0.1       -> 127.0.0.1      : GET / HTTP/1.1

12/31 23:45:51 127.0.0.1       -> 127.0.0.1      : GET / HTTP/1.1

12/31 23:45:51 127.0.0.1       -> 127.0.0.1      : GET / HTTP/1.1

12/31 23:45:51 127.0.0.1       -> 127.0.0.1      : GET / HTTP/1.1

12/31 23:45:51 127.0.0.1       -> 127.0.0.1      : GET / HTTP/1.1

12/31 23:45:51 127.0.0.1       -> 127.0.0.1      : GET / HTTP/1.1

12/31 23:45:51 127.0.0.1       -> 127.0.0.1      : GET / HTTP/1.1

12/31 23:45:51 127.0.0.1       -> 127.0.0.1      : GET / HTTP/1.1
```

3. Le programme commence d'abord par initialiser les valeurs par défaut si aucun argument n'est donné au lancement de websnarf, ensuite pour chaque argument s'il est donné on affiche/effectue l'action correspondante (affichage de l'aide, changement du port d'écoute, timeout, fichier pour les logs, format apache ou non…)

Le programme créé un socket d'écoute sur le port demandé, si la création de ce socket échoue on affiche une erreur sinon on initialise la variable timeout à 0 et le client en « non défini »

On boucle ensuite lorsqu'un client se connecte et on enregistre tout de suite les adresses distante et locale avant de lire la requête, chaque étape est détaillée lorsqu'on ajoute l'argument « --debug ».

Pour la lecture de la requête on compte le nombre de boucles de lecture et on s'arrête si il y a un timeout ou si la requête est terminée. Après la fin de la lecture on vérifie le nombre de boucles et la taille de la requête, si il y a 0 boucles, on affiche le timeout, si la taille est de 0 on affiche que la requête est vide. Sinon, on enregistre toutes les informations de la connexion dans une liste et on affiche/enregistre dans le fichier de log si demandé dans le format renseigné au préalable.

4. Le programme fonctionne de la même façon que l'original et l'affichage est identique, il s'utilise de la manière suivante :

```
joran@ubuntu:~/Documents/M1/Reseaux/Projet$ ./websnarf -h
usage: ./websnarf [options]
--timeout <n> or -t <n>    wait at most <n> seconds on a read (default $alarmtime)
--log <FILE> or -l <FILE>  append output to FILE (default stdout only)
--port <n> or -p <n>       listen on TCP port <n> (default $port/tcp)
--max <n> or -m <n>        save at most <n> chars of request (default $maxline chars)
--savedir <DIR> or -s <DIR>   save all incoming headers into DIR
--debug or -d              turn on a bit of debugging (mainly for developers)
--apache or -a             logs are in Apache style
--fullnames or -f          display fullname instead of IP adress
--help or -h               show this listing
```

5. Open issues :

Le premier « problème » est le manque de reverse DNS, c'est à dire que le programme ne peut pas trouver un nom à partir d'une IP, cela peut permettre de voir si l'adresse appartient à un domaine que l'on veut bloquer par exemple.

Ensuite, la gestion du timeout de websnarf est un peu bancale, elle peut poser problème si le client se déconnecte de façon inattendue par exemple, la connexion pourrait bloquer.

Il n'a pas vraiment de capture de paquet, juste une partie de la requête est affichée ce qui ne permet pas d'avoir toutes les informations contenues dans celle-ci.

Le programme ne gère pas les connections concurrentes, il attend qu'un connexion soit terminée avant de passer à une autre. Si beaucoup de requêtes sont faites, une connexion peut bloquer les autres et cela augmente aussi le temps de process.

Le programme n'est pas un « daemon » c'est à dire qu'il ne s'exécute pas en arrière-plan, quelqu'un doit surveiller les requêtes manuellement. Cela est réglable quand on sait quelles requêtes on cherche et en créant des processus fils qui vont s'occuper de chaque connexion en arrière-plan.

Le programme ne distingue pas les versions différentes de Code Red.

Le programme ne propose pas les logs dans le format ISS, ce format affiche notamment le fichier demandé par la requête et le temps de process de la requête en plus des autres informations comme

l’adresse, la date, l’heure etc. Tous les champs sont séparés par des virgules. Pour ajouter ceci, il suffit de parser la requête et de compter le temps de process pour l’afficher ensuite.

Security Issues :

Le programme ne comporte pas vraiment de problème au niveau de la sécurité de la machine, il démarre pas de sous-programmes et ne comporte pas de code dangereux/virus.

Par contre, il ne détecte pas la variante Code Red 2.

Fonctionnalités ajoutées et fixes pour certains bugs :

-f ou –fullnames : affiche les noms plutôt que les IP

- Affichage du port local et distant en plus des adresses

-Si le dossier renseigné avec l’option « savedir » n’existe pas, il est automatiquement créé avec les droits « rwx » pour l’utilisateur.

-Lors des logs apache la timezone est réellement affichée au lieu de simplement -0000 dans le programme de base.

- Lors des logs apache la vraie taille de la requête est affichée au lieu de 100 dans le programme de base

- Correction d’un bug lors du mode « debug » qui faisait tourner la lecture du message en boucle si un client se déconnecte de manière inattendue. (testé avec netcat).

- Connexion concurrentes : à chaque connexion un processus fils est créé permettant de s’occuper de plusieurs connexions à la fois. Cela serait aussi possible en remplacent les forks par des threads.

Le programme fonctionne sur les machines Linux et Windows (non testé sur MacOs) et ne demande pas de librairies spéciales pour fonctionner.