Sanson Antoine M1 CyberSécurité
Delsol Benjamin M1 CyberSécurité

04/05/2022

WireFish

Introduction

- Nombreuses fonctionnalités : **recherche/tri/suivi** de flux de réseaux mais aussi le **déchiffrement** de certains protocoles, la décompression des données à la volée
- Opensource
- Logiciel très populaire
- Point négatif : Affichage peu lisible ainsi de nombreuses informations exhaustives

Cas d'utilisation

Test d'intrusion

- Aucun trafic émanant de WireFish

- Problème environnemental (interpréteur Python3, connexion internet) ou avec un exécutable avoir les droits administrateur sur la machine

Cas d'utilisation

Man In The Middle

- Programme d'écoute exécuté sur la machine de l'attaquant

État de l'Art

- Utilitaire ligne de commande
- Affiche l'ensemble des interfaces réseau, capture le trafic d'une interface, applique des filtres plus ou moins complexes et affiche à l'écran ou bien enregistre les paquets dans un fichier de capture réseau

NetworkMiner

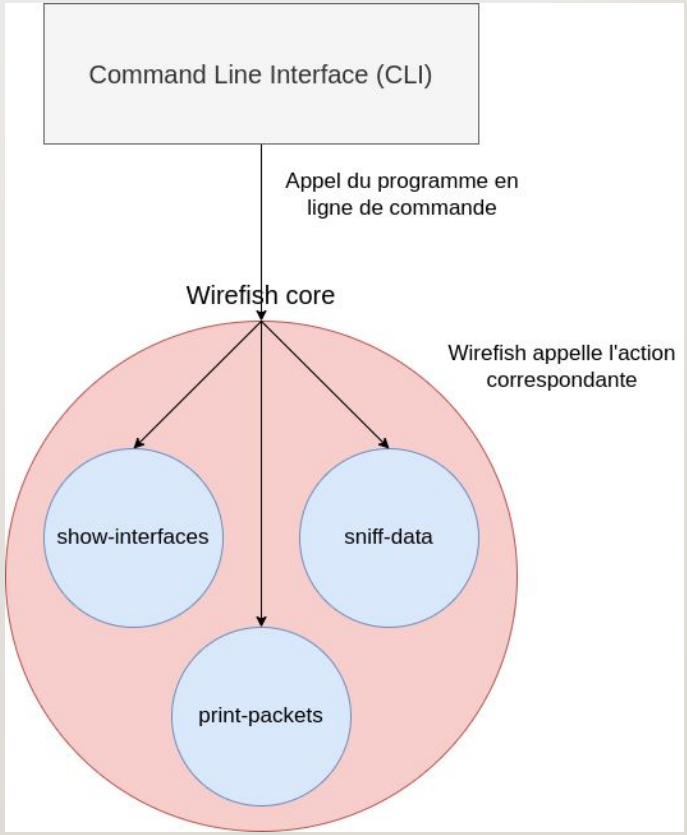
- Permet de détecter par exemple le systèmes d'exploitation, noms d'hôtes, ports ouverts, et bien d'autres
- Supporte de nombreux protocols applicatifs comme **FTP**, **HTTP**, **SMB**, **SMTP**

Fonctionnement de WireFish

- show-interfaces : affiche l'ensemble des interfaces réseau disponibles, avec des détails comme l'adresse MAC ou l'adresse IPv4 associée.

- print-packets : permet d'écouter une ou plusieurs interfaces et affiche des informations élémentaires sur les paquets en ligne de console (filtrage possible)

- sniff-data : permet d'analyser le trafic en temps réel et d'en extraire les données sensibles à l'aide des modules développés



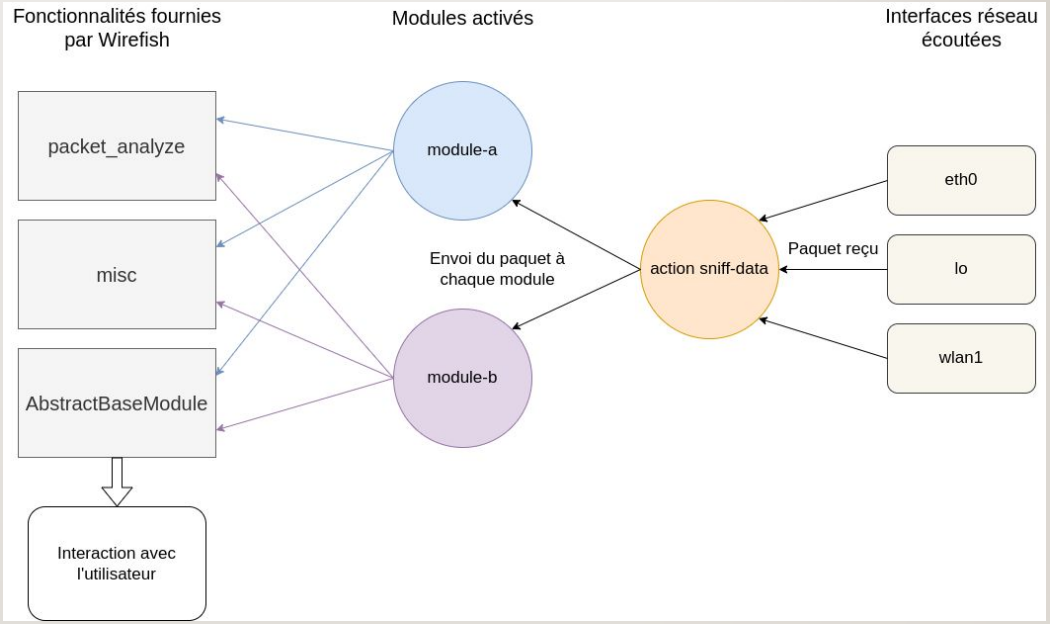
La commande Sniff-data

- packet_analyze : permet d'analyser et d'extraire des données ou d'interpréter le payload de certains paquets.

- misc : contient des fonctions utilitaires mais qui n'ont pas de lien direct avec le traitement des paquets.

- AbstractBaseModule est la mère de tous les modules, elle fournit des méthodes et des propriétés utiles aux classes dérivées (modules créés).

  Chaque module doit implémenter correctement les éventuels champs abstraits de cette classe, sans quoi le module ne sera pas chargé et donc inutilisable. Elle fournit au passage des fonctionnalités de logging et d' écriture de fichiers.



Documentation : Installation

Prérequis :

- Avoir installé Python3

- Récupérer le code sur GitHub
https://github.com/Schrubitteflau/WireFish/

Démarche

- Lancer le script d'installation **setup.sh**

- Sous Windows, vérifier la création de l'environnement virtuel

Documentation : Fonctionnelle

```
$> ./main.py
usage: main.py [-h] {show-interfaces,print-packets,sniff-data} ...
main.py: error: the following arguments are required: command_action
```

Utilisation de **argparse** assurer la viabilité des paramètres

Documentation : Fonctionnelle

```
$> ./main.py sniff-data -I wlp164s0,lo -M http.post_credentials,ftp.credentials,ftp.transfer_files
Sniffing interface(s) ['wlp164s0', 'lo'] using filter '<no filter>' and module(s) ['http.post_credentials', 'ftp.credentials', 'ftp.transfer_files']
Loading module http.post_credentials : Success
Loading module ftp.credentials : Can't instantiate abstract class Module with abstract methods on_receive_packet
Loading module ftp.transfer_files : Success
```

-I précise les interfaces

-M précise les modules

Présentation du module : **ftp.credentials**

```
ftp_packet_parser  = FTPPacketParser(packet=packet)
if ftp_packet_parser.is_command("USER"):
    self.log_message("FTP username :  %s " % (

self.to_str_safe(ftp_packet_parser.get_parameters())
))
```

```
00:03:10 - info - ftp.credentials : FTP username : ftpuser
00:03:10 - info - ftp.credentials : FTP password : passw0rd
```

Il intercepte simplement les noms d'utilisateurs et mots de passe transmis d'un client vers un serveur, qui sont sous la forme de commandes **USER** *<username>*, et **PASS** *<password>*

Présentation du module : **ftp.transfer_files**

```
00:18:16 - info - ftp.transfer_files : RETR confidentiel
00:18:16 - info - ftp.transfer_files : RETR confidentiel : written sniff_files/ftp.transfer_files/1651529896_confidentiel (23 bytes)
00:18:25 - info - ftp.transfer_files : STOR WireFish.pdf
00:18:25 - info - ftp.transfer_files : STOR WireFish.pdf : written sniff_files/ftp.transfer_files/1651529905_WireFish.pdf (269985 bytes)
```

- Le client envoie une requête **PASV** afin d'ouvrir une connexion passive
- Le serveur donne une IP et un port pour la connexion
- Le client initie la connexion en arrière plan
- Le client effectue une requête FTP (sur le port d'écoute du serveur) qui nécessite un transfert de données.
- Le serveur va envoyer les données à travers la connexion passive précédemment établie
- Le client et le serveur peuvent toujours communiquer sur le port du serveur **FTP**, et le serveur indique au client lorsqu'il a terminé d'envoyer les données. La connexion passive est alors coupée et le client a téléchargé le fichier.

Présentation du module : **http.post_credentials**

```
00:43:42 - info - http.post_credentials : GET localhost:8080/, Authorization header : <no-data>, Cookie header : <no-data>
00:43:42 - info - http.post_credentials : Response code : 302, Set-Cookie header : <no-data>
00:43:42 - info - http.post_credentials : GET localhost:8080/login, Authorization header : <no-data>, Cookie header : <no-data>
00:43:42 - info - http.post_credentials : Response code : 200, Set-Cookie header : <no-data>
00:43:53 - info - http.post_credentials : POST localhost:8080/login, Authorization header : <no-data>, Cookie header : <no-data>
00:43:53 - info - http.post_credentials : b'username=super_username&password=passw0rd'
00:43:53 - info - http.post_credentials : Response code : 302, Set-Cookie header : sessionid=c78d314b-59fa-4e69-a78c-f93e991cfbf6; Path=/
00:43:53 - info - http.post_credentials : GET localhost:8080/dashboard, Authorization header : <no-data>, Cookie header : sessionid=c78d314b-59fa-4e69-a78c-f93e991cfbf6
```

Le but est de rechercher dans les paquets les requêtes et réponses HTTP

Dans un premier temps, l'utilisateur accède à la racine, mais il n'est pas connecté donc est redirigé vers **/login**. Il remplit et soumet un formulaire via la méthode **POST**, on voit que son nom d'utilisateur est **super_username** et que son mot de passe est **passw0rd**. Le serveur redirige l'utilisateur vers **/dashboard** en lui envoyant un cookie de session **sessionid**.

Présentation du module : **http.post_credentials**

```
def _guess_payload_class(self, payload) -> PayloadClass:
    """ Decides if the payload is an HTTP Request or Response, or something else
"""
    try:
        crlfIndex = payload.index("\r\n".encode())
        req = payload[:crlfIndex].decode("utf-8")
            if http_request_regex.match(req):
                return PayloadClass.HTTP_REQUEST
            elif http_response_regex.match(req):
                return PayloadClass.HTTP_RESPONSE
    except:
        pass
        return PayloadClass.UNKNOWN
```

L'analyse des requêtes et réponses HTTP est nativement présente avec **Scapy**

On obtient des instances de *HTTPRequest* ou *HTTPResponse*, et accéder à des propriétés comme **request.Authorization**, **request.Host**, **response.Set_Cookie**,

Présentation du module : **http.download_files**

```
01:08:06 - info - http.post_credentials : GET localhost:8080/dashboard, Authorization header : <no-data>, Cookie header : sessionid=c78d314b-59fa-4e69-a78
c-f93e991cfbf6
01:08:06 - info - http.post_credentials : Response code : 304, Set-Cookie header : <no-data>
01:08:08 - info - http.post_credentials : GET localhost:8080/uploads/confidentiel, Authorization header : <no-data>, Cookie header : sessionid=c78d314b-59
fa-4e69-a78c-f93e991cfbf6
01:08:08 - info - http.download_files : Written file of type application/octet-stream as sniff_files/http.download_files/1651532888_tuutecoqpw (23 bytes)
01:08:08 - info - http.post_credentials : Response code : 200, Set-Cookie header : <no-data>
```

Permet de conserver une copie locale des fichiers téléchargés par le client.

L'utilisateur accède à **/dashboard**, puis il télécharge un fichier **/uploads/confidentiel**. Celui-ci est intercepté par ce *ftp.download_files* et est enregistré localement sous le nom de *1651532888_tuutecoqpw*, qui est la combinaison du timestamp et d'une chaîne aléatoire.

Malheureusement le module ne fonctionne pas pour les fichiers transmis sur différents paquets par soucis de taille.

Mais aussi le lien entre la requête et la réponse n'est pas effectué donc la certitude de la source du fichier n'est pas garantie

Choix des technologies

- Exécution facile des programmes
- Syntaxe simple et concise
- Grandes quantités de bibliothèque

- Regroupe des fonctions sur le traitement des paquets
- Simple d'utilisation
- Créée par un ingénieur Français **Philippe Biondi**

Difficultés Rencontrées

Scapy problèmes de performance ⇒ Résultats incohérents ou perte de données

<u>Solution partielle:</u>

Utilisation de **libpcap,** programme en C pour capturer les paquets, plus performants

⇓

Contrainte supplémentaire dans l'environnement

Difficultés Rencontrées

Difficultés à interpréter différents payloads à la suite (exemple transfert de fichiers), un port peut servir seulement pour un transfert précis après la connexion sera coupée

Solution :

Être capable de comprendre le protocole analysé et programmer ce cas précis

Difficultés Rencontrées

Limites d'un module :

Le développement de module avançant sur un protocole précis → "client muet" → Problème de lisibilité du code, fonctionnalités de base ne suffisent plus

Axes d'Amélioration

- Exporter dans un fichier les log
- Ajouter la prise en charge d'une configuration externe pour rendre les modules paramétrables (ex: limite fichier sauvegarde)
- Désolidariser les modules en fonction d'un protocole ou d'une IP = code plus clair + de performance
- Prise en charge d'un fichier de capture réseau

Conclusion - Apports Personnels

Démonstration