Hénallux
Section sécurité des systèmes

Développement

Implémentation d'un IDS

Projet réalisé par

Mustafa-Can KUS
Jordan DALCQ

Année académique 2020-2021

Contents

Part I

Introduction

Chapter 1

Contexte

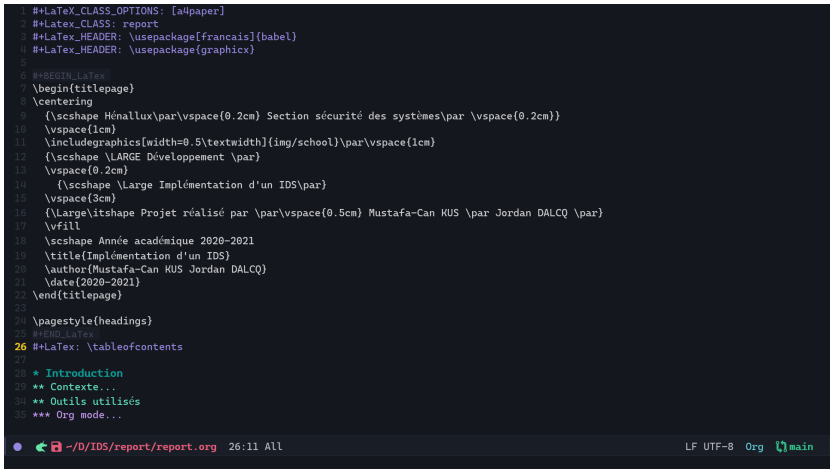
Dans le cadre du cours de développement il nous a été demandé de réaliser un système de détection d'intrusion (ou IDS pour faire plus court). Ce programme à pour but d'analyser le trafic réseau et reporter toutes activité suspectes à un administrateur système ou même un administrateur réseau grâce à un système de log.

Chapter 2

Outils utilisés

2.1 Org mode

Org mode est un mode majeur pour le logiciel GNU Emacs qui permet de prendre note et maintenir une "To do list" et permet aussi de planifier facilement des projets grâce à son langage Markup (très proche du markdown). On l'a utilisé pour écrire ce rapport (ok on a un peu triché on a mit un peu de Latex pour faire joli) et aussi pour planifier notre travail



2.2 Github

Une fois notre planning fait il nous fallait une solution pour que chacun d'entre nous dispose d'une copie du code à portée de main et qu'on puisse tracker nos modifications, ce qui est très pratique en cas de bug, en effet il nous suffit de revenir quelques modifications en arrière et le problème est réglé !

Notre projet est disponible ici:

`https://github.com/Les-IRaniens/IDS`

2.3 GNU Makefile

Comme tout bon informaticien qui se respecte, on a pas envie de taper une très longue commande composé d'une dizaine de fichiers et d'une autres dizaine de flag à chaque fois qu'on souhaite compiler notre programme, alors on a décidé d'utiliser un makefile. Le makefile s'occupe de la compilation et de linker notre code automatiquement, il suffit de taper make dans le terminal et le tour est joué !

2.4 Nos flags de compilations

- -pednatic: nous oblige fortement à adhérer aux règles de l'ANSI C
- -Wpednatic: nous affiche des warnings si on respecte pas la pedantique
- -Wall: nous permet d'avoir tous les warnings sur des pratiques considérées comme questionnable
- -Wextra: Couvre encore plus de warnings que -Wall
- -Werror: Transforme tous les warnings en erreur (Oui on est sans pitié ici)
- -g: Permet d'avoir les symboles de debuggage
- -Isrc/: Permet d'inclure facilement les headers du dossier src
- -fsanitize=undefined & -fsanitize=undefined: Permet de tracker chaque memory leak et nous dit sur quelle ligne est le problème
- -lpcap: Inclut la libpcap à notre projet.

2.5 ArchLinux

Programmer sur Kali c'est pas top, surtout sur une VM ! Donc on a préféré utiliser ArchLinux pour le travail sur machine native. Pourquoi cela ? Car Archlinux est une distribution polyvalante qui a TOUS les paquets qu'on désire (oui même tous les paquets de Kali).

2.6 Vscodium

C'est Visual Studio Code - les fonctions de télémétries, c'est notre éditeur de choix, car il permet une super bonne intégration avec notre github, nous informe de nos erreurs et des warnings potentiels grâce à l'extension C/C++.

Part II

Explications du code

Dés le départ nous avons pensé à subdiviser le code en plusieurs fichiers, cela permet de limiter le nombre de ligne de code, en effet le maximum par fichier est de 300 lignes de code, ce qui permet de trouver nos erreurs plus efficacement.

Chapter 3

main.c

3.1 int main (int argc, char *argv[])

C'est un peux le chef d'orchestre de notre programme, ce fichier contient la fonction main qui donne les entrées utilisateurs au différentes fonctions grâce aux paramètres donné au programme (argv) et vérifie que l'utilisateur démarre bien le programme avec \.

Chapter 4

usage.c

4.1 void show_usage(char *exe_path

) Si l'utilisateur nous donne aucun paramètre ou si il nous donne le paramètre d'aide (-h | –help) la fonction main nous envoie sur cette fonction et nous montre un message d'utilisation. L'argument exe_path comprend le chemin de l'executable et on récupère que le nom de l'executable (grâce à la fonction basename).

Chapter 5

checkuser.c

5.1 bool is_root(void)

Il s'agit d'une simple vérification pour voir si on démarre bien le programme en tant qu'administrateur système.

Chapter 6

utils/list.c

6.1 void init_str_list(StrList *self)

Cette fonction est le constructeur de notre objet StrList (il s'agit d'une structure qui correspond à un tableau de string alloué de manière automatique) Il attribue un emplacement mémoire de base.

6.2 void append_str

_list(StrList *self, const char *str) La fonction vérifie tout d'abord si notre liste à assez de place en mémoire, si ce n'est pas le cas on lui donne plus d'emplacement mémoire et on ajoute la chaîne de caractère donné en paramètre à la fin de la liste.

6.3 StrList split_to_liststr(char *str, const char *del)

Cette méthode découpe une chaîne de caractère selon un délimiteur et met chaque partie dans une liste.

6.4 void free_str_list(StrList *self)

Cette fonction libère la mémoire occupée par une liste.

Chapter 7

utils/log.c

7.1 void log_ids(const char *msg)

Cette fonction écrit dans le syslog.

Chapter 8

readrules.c

8.1 StrList read_rules(const char *filename)

C'est ici que se produit la lecture des règles (données en paramètre par l'utilisateur), le fichier est lu ligne par ligne et placé dans une liste de string.

Chapter 9

rule.c

9.1 RuleList parse_rule(StrList rules)

Voilà la fonction qui est responsable de traduire chaque règle en une structure qu'on a simplement nommé \, qu'on place dans une liste qu'on a nommé \ qui occupe un certain emplacement mémoire; la struct rule contient le protocole, l'addresse ip de source et de destination, le protocole de source et de destination.

9.2 bool is_in_context(Rule rule, void *packet, Protocole proto)

Cette fonction vérifie si le paquet correspond avec une des règles.

9.3 free_rules(RuleList *lst)

cette fonction libère l'emplacement mémoire pris par les listes de règles.

Chapter 10

scan.c

10.1 void scan_network(char *interface, RuleList rules)

Cette fonction permet de préparer la libpcap et le scan du réseau.

10.2 void handler(u_char *user, const struct pcap_pkthdr *header, const u_char *packet)

Cette fonction est le point central de notre IDS, elle capture et analyse chaque paquet qui passe par le réseau, elle utilise la fonction populate (présente dans le fichier populate.c) pour convertir les données binaires reçues dans le réseau vers des structures plus simples à travailler. Si une règle correspond ; il donne un message prédéfini dans le dossier des règles au système de log de l'IDS.

Part III

Critère de dépassement

Voici une petite liste des petits plus qu'on a ajouté au projet

- Utilisation des services Linux pour faire tourner l'application en tâche de fond (make install).
- Construire une règle permettant de détecter une attaque de type XSS (simple). Cette règle devra être utilisable dans votre IDS :

Pour cela nous avons ajouté un nouveau paramètre dans les règles nommé client-side-content, ce paramètre se comporte comme content mais s'occupe que du côté client et ignore le côté serveur.

- Gestion des versions (Github).
- On a un Makefile (Ca compte comme bonus non ?).