Projet C:
Implémentation d'un *IDS*

Adrien Voisin, Didier Valentin, Bastien Bodart
Développement - IR209

Henallux - 2020-2021

1 Introduction

Dans le cadre du cours *IR209: Développement*, il vous est demandé d'implémenter un IDS[1] simplifié en C. Un IDS (Intrusion Dectection System) est un système permettant de détecter des activités suspectes sur une machine ou sur le réseau. Si une menace est identifiée, alors l'IDS déclenchera une alerte.

Pour votre projet, on vous demande de réaliser un IDS capable d'écouter le trafic d'une interface réseau sur une machine virtuelle en utilisant la librairie libpcap[2]. Pour chaque paquet que vous considérez comme suspect, votre programme ajoutera une entrée dans syslog[3].

2 Architecture du système

Pour tester votre système, il vous faudra utiliser deux machines virtuelles:

- VM Kali Linux: votre programme tournera sur cette machine et écoutera le trafic qui transite par son interface réseau.
- VM Debian Linux: cette machine sera utilisée pour vos tests. On y retrouvera une série de services comme un serveur web, un serveur ftp etc.

  Ces deux machines virtuelles seront configurées avec une interface interne permettant d'éviter tout trafic parasite. L'objectif de la machine Debian est de générer du trafic divers (HTTP, TCP, UDP etc).

[1] https://en.wikipedia.org/wiki/Intrusion_detection_system
[2] https://github.com/the-tcpdump-group/libpcap
[3] https://www.gnu.org/software/libc/manual/html_node/Syslog-Example.html

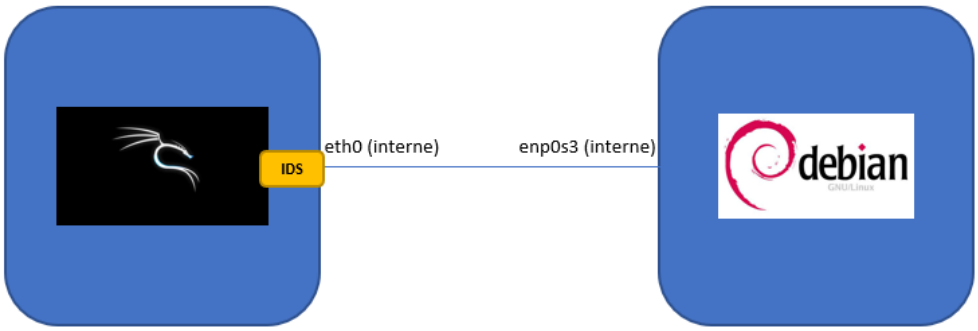

Figure 1: Architecture du système.

3 Architecture du logiciel

L'objectif de votre programme sera de récupérer une copie de chaque paquet transitant par le réseau. Les informations contenues dans ce paquet seront stockées dans une structure. Il faudra ensuite confronter ces données avec une liste de règles se trouvant dans un fichier. Si les caractéristiques ou le contenu d'un paquet sont considérés comme suspects par l'une des règles se trouvant dans le fichier, alors il faudra journaliser le message correspondant dans syslog. Voici un schéma reprenant les principaux éléments de votre programme:

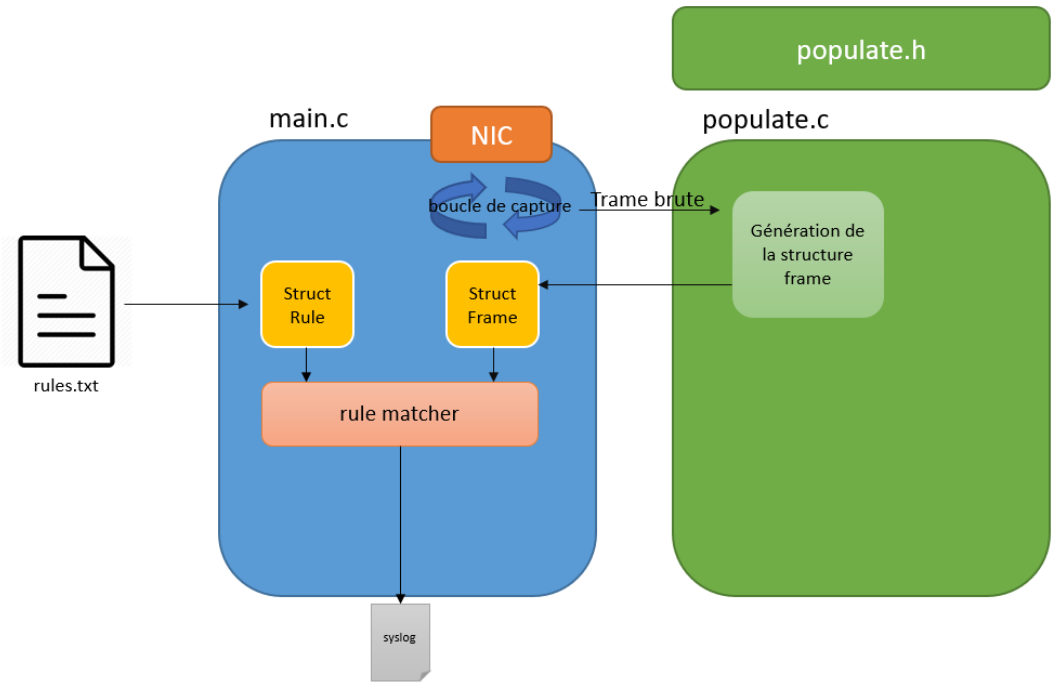

Figure 2: Architecture logicielle.

3.1 Les règles

Un IDS repose généralement sur une liste de règles pour déterminer si un paquet réseau représente une menace. Dans votre système, ces règles seront stockées dans un fichier texte simple. Le format de ces règles devra respecter le formalisme suivant:

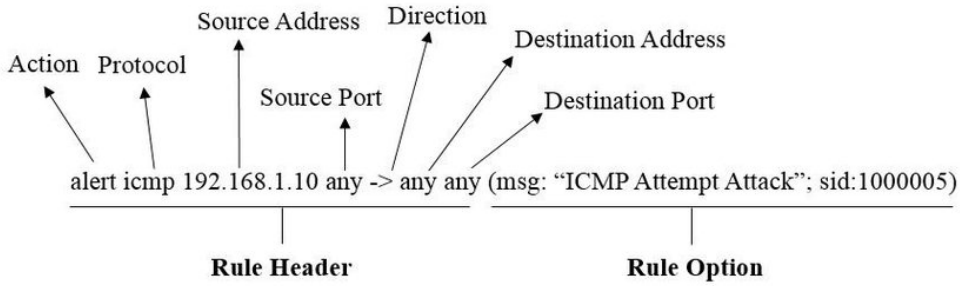

Figure 3: Format d'une règle.

- Action: détermine l'action à prendre pour un paquet qui "*match*". Dans le cadre de ce projet on se limitera à générer une alerte dans syslog.
- Protocole: protocole utilisé par le paquet. Celui-ci peut correspondre à un protocole de transport (TCP, UDP) ou à un protocole applicatif (HTTP, FTP, ICMP...). Pour le projet, on vous demande de traiter au minimum TCP, UDP et HTTP
- Adresse source: correspond à l'adresse de la machine qui envoie le paquet.
- Port source: numéro de port utilisé pour envoyer le paquet
- Direction: donne une indication à l'IDS sur la manière de matcher un paquet. Dans le contexte du projet, on pourra ignorer ce paramètre.
- Adresse destination: correspond à l'adresse de la machine qui reçoit le paquet.
- Port destination: numéro de port utilisé pour la réception du paquet
- Options: les options doivent être spécifiées entre des parenthèses. Chaque option est représentée par une clé:valeur. De plus, chaque option est séparée par un ';'. Ces options seront utilisées différemment en fonction de l'IDS. Pour le projet, on vous demande d'utiliser au minimum l'option 'content' et 'msg'. Le contenu permet de matcher une chaine de caractère avec le contenu d'un paquet. Le message lui sera utilisé pour générer une entrée dans syslog.

Exemple: imaginons que notre fichier contienne deux règles:

- alert http any any $->$ any any (msg:"shell attack"; content:"malware.exe";)
- alert tcp any any $->$ any 8888 (msg:"backdoor attack";)

Si un site Internet est consulté et que dans la page on retrouve la suite de caractères "malware.exe", alors, votre IDS devra ajouter dans syslog une alerte contenant le message "shell attack". Remarquez qu'il n'y a pas de limitations sur les adresses ou numéros de ports mais seulement sur le protocole (http). Dans le deuxième exemple, si une connexion TCP sortante pointe vers le port 8888, alors un message "backdoor attack" devra être ajouté au syslog. Pour le projet, un fichier de règle type vous sera fourni. N'hésitez pas à ajouter des règles si vous voulez tester la résilience de votre IDS. Ces règles sont basées sur les formats utilisés sur des IDS comme Snort[4] ou Suricata[5]

3.2 Le fichier populate

Le fichier populate.c permet de récupérer une trame Ethernet depuis l'interface réseau et de remplir une structure utilisable facilement dans votre fichier main. La trame est tout d'abord castée à l'aide des différentes structures vues au cours et ensuite les informations clés sont placées dans une structure Frame utilisable dans votre code.

```
struct custom_udp
{
        int source_port;
        int destination_port;
        unsigned char *data;

} typedef UDP_Packet;

struct custom_tcp
{
        int source_port;
        int destination_port;
        int sequence_number;
        int ack_number;
        int th_flag;
        unsigned char *data;
        int data_length;

} typedef TCP_Segment;

struct custom_ip
{
        char *source_ip;
        char *destination_ip;
        TCP_Segment data;

} typedef IP_Packet;
```

[4]https://www.snort.org/
[5]https://suricata.readthedocs.io/en/latest/rules/intro.html

```
struct custom_ethernet
{
        char *source_mac;
        char *destination_mac;
        int ethernet_type;
        int frame_size;
        IP_Packet data;

} typedef ETHER_Frame;
```

Pour générer cette structure à partir d'une trame brute, vous pouvez appeler la fonction *populate_packet_ds()* dont la signature est la suivante:

```
int populate_packet_ds(const struct pcap_pkthdr *header,
        const u_char *packet, ETHER_Frame *frame);
```

Cette structure sera garnie avec les informations minimum requises pour vérifier le contenu d'un paquet. Libre à vous de modifier ce fichier pour y ajouter d'autres informations.

3.3 Le rule matcher

Le *rule matcher* correspond à une fonction que vous allez devoir implémenter. Elle prendra comme paramètre la structure contenant les informations d'une trame (ETHER_Frame) et la structure représentant les règles lues dans le fichier texte. La fonction devra vérifier si le paquet "match" avec l'une des règles. Si c'est le cas, alors on journalisera un message dans syslog.

4 Ce que vous allez devoir faire

Au démarrage du projet, vous allez recevoir un squelette du programme. Celui-ci sera composé des fichiers main.c, populate.c et populate.h. Vous pouvez compilez ce programme et le tester à l'aide de la commande:

```
gcc -Wall -o ids main.c populate.c -lpcap
```

Attention d'utiliser sudo pour permettre à libpcap de fonctionner. A parti de cela, on vous demande d'implémenter au minimum les fonctions suivantes:

```
void read_rules(FILE * file, Rule *rules_ds, int count)
void rules_matcher(Rule *rules_ds, ETHER_frame *frame);
```

Ces fonctions peuvent évidemment en appeler d'autres. Libre à vous d'organiser votre code selon vos besoins. De plus, vous pouvez modifier le squelette et le fichier populate.c si cela vous semble nécessaire (attention toutefois à d'abord se concentrer sur les objectifs de base). Attention de ne pas sous-estimer le temps nécessaire pour implémenter ces deux fonctions.

4.1 Read rules

Cette fonction permettra de lire les entrées du fichier rules.txt et de les placer dans une structure qui vous semble adéquate. Veillez à bien réfléchir si ces informations peuvent être comparées à celles se trouvant dans ETHER_Frame. Afin de tester cette fonction, un fichier de base sera mis à disposition. La fonction reçoit un pointeur vers un fichier déjà ouvert, un pointeur vers la structure à garnir ainsi que le nombre de règles présentes dans le fichier (à vous de le déterminer). Par défaut la fonction ne revoie rien. Toutefois, si nécessaire, vous pouvez utiliser la valeur de retour pour envoyer un code d'erreur (int). **Attention, le nom du fichier devra être renseigné à l'exécution, en utilisant les arguments (argv[]).**

4.2 Rule matcher

Pour chaque paquet reçu et garni, il sera nécessaire de le comparer avec vos règles afin de déterminer si le paquet est suspect ou non. C'est cette fonction qui écrira dans syslog si nécessaire. Le fonction reçoit comme paramètre un pointeur vers la structure contenant les règles ainsi qu'un pointeur vers la structure contenant les données d'un paquet. Par défaut la fonction ne revoie rien. Toutefois, si nécessaire, vous pouvez utiliser la valeur de retour pour envoyer un code d'erreur (int).

5 Evaluation

5.1 Acquis d'apprentissage

L'évaluation a pour objectif de valider les acquis d'apprentissage tels que décrits dans la fiche UE du cours:

- Développer une application C dans le contexte de la sécurité informatique
- Comprendre les enjeux de la gestion de la mémoire en C
- Argumenter les choix d'implémentation

5.2 Critères minimum de réussite

Votre système devra au minimum respecter les contraintes suivantes

- Programme qui compile et fonctionne sans erreurs
- Respect des consignes de l'énoncé
- Respect des bonnes pratiques vues durant les cours et laboratoires
- Le programme est capable de détecter du trafic malveillant en fonction du fichier de règles fourni aux étudiants
- Utilisation des arguments pour renseigner le nom du fichier de règles (argv)

- Détection de l'encryption du payload
- Respect de la deadline

5.3 Critères de dépassement

Une fois les critères de base rencontrés, libre aux étudiants de dépasser ces fonctionnalités et d'ajouter une touche personnelle à leur système. Ci-dessous une liste non exhaustive de critères de dépassement:

- Gestion des attaques de type SYN Flood
- Gestion d'un grand nombre de protocoles
- Utilisation des services Linux pour faire tourner l'application en tache de fond[6]
- Construire une règle permettant de détecter une attaque de type XSS (simple). Cette règle devra être utilisable dans votre IDS
- Gestion des versions [7]
- Originalité
- ...

6 Modalités pratiques

- Le projet doit être réalisé par groupes de deux étudiants.
- Les étudiants devront être capables de défendre chaque partie du système.
- De plus, les étudiants seront interrogés de manière individuelle sur des questions théoriques liées au cours.
- Toute tentative de triche ou de plagiat sera sanctionnée d'un zéro.
- L'examen se déroulera à distance. Les étudiants utiliseront deux machines virtuelles pour faire une démonstration de leur système en partageant leur écran. Les inscriptions se feront via Moodle.

[6] https://www.howtogeek.com/687970/how-to-run-a-linux-program-at-startup-with-systemd/
[7] https://github.com/

7 Délivrable

Pour le 03/01/2021 à 23h59, les étudiants devront remettre un délivrable contenant les éléments suivants (un dépôt par groupe):

- Le code source de votre programme
- Référence éventuelle à un dépôt *Git*
- Une documentation aussi complète que possible expliquant le fonctionnement et les choix d'implémentation du système. Cette documentation fera office de rapport pour le projet.

8 Le plagiat

Attention de toujours bien indiquer vos références concernant les parties de code dont vous n'êtes pas l'auteur[8].

9 Contact

En cas de problème/questions contactez en priorité le professeur en charge de votre laboratoire. Si nécessaire le professeur responsable de l'UE: adrien.voisin@henallux.be

[8] https://fr.wikipedia.org/wiki/Plagiat