Projet de développement - IR209 (Implémentation d'un IDS en C)

Aazzani Bilal

Sécurité des systèmes - IRB2

Adrien Voisin, Didier Valentin, Bastien Bodart

13/08/2021

Définition d’un ids :

Un système de détection d'intrusions (« **Intrusion Detection** Systems » ou **IDS**) est un appareil ou une application qui alerte l'administrateur en cas de faille de sécurité, de violation de **règles** ou d'autres problèmes susceptibles de compromettre son réseau informatique.

https://www.lemagit.fr/definition/Systeme-de-detection-dintrusions

Contenu du fichier de règles :

alert ftp any any -> any any (msg:"Unsecure protocol use detected";)

alert tcp any any -> any 8888 (msg:"backdoor attack";)

alert dhcp any any -> any any (msg:"dhcp spoofing";)

alert udp any any -> any 9999 (msg:"backdoor attack";)

Commande pour compiler le programme :

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

Commande pour exécuter le programme :

./ids ids.rules

main.c

- **Struct ids_rule (typedef Rule)**

  Cette structure est utilisée pour représenter une règle présente dans le fichier de règles (par défaut ids.rule). Cette structure est composée de tous les éléments dont nous avons besoin dans la règle; le nom du protocole, l'adresse ip et le port de la source et de la destination, ainsi que le message qui serait affiché dans le syslog. Il y a également un élément pour le contenu, qui peut être utilisé dans certaines règles.

- **Struct configuration (typedef Configuration)**

  Cette structure est utilisée pour pouvoir facilement passer en argument à la fonction pcap_loop les paramètres que nous avons besoin, c'est-à-dire un pointeur vers les règles et le nombre de règles que nous avons.

- **Bool check_protocole(Rule* rule, int transport_protocol, char* application_protocol)**

  Cette fonction vérifie si le protocole défini dans une règle (rule) match avec le protocol (de transport ou d'application) utilisé dans une frame. Si c'est le cas, cette fonction retourne true, sinon false.

- **Bool check_source_ip (Rule* rule,char* source_ip)**

  Cette fonction vérifie si l'adresse ip de la source définie dans une règle (rule) match avec l'adresse ip de la source d'une frame capturée. Si c'est le cas, cette fonction retourne true, sinon false.

- **Bool check_destination_ip(Rule* rule,char* destination_ip)**

  Cette fonction vérifie si l'adresse ip de la destination définie dans une règle (rule) match avec l'adresse ip de la destination d'une frame capturée. Si c'est le cas, cette fonction retourne true, sinon false.

- **Bool check_source_port (Rule* rule,int source_port)**

  Cette fonction vérifie si le port de la source défini dans une règle (rule) match avec le port de la source d'une frame capturée. Si c'est le cas, cette fonction retourne true, sinon false.

- **Bool check_destination_port(Rule* rule,int source_port)**

  Cette fonction vérifie si le port de la destination défini dans une règle (rule) match avec le port de la destination d'une frame capturée. Si c'est le cas, cette fonction retourne true, sinon false.

- **Bool check_content (Rule* rule, char* payload)**

  Cette fonction vérifie si une règle (rule) définit une contrainte sur le contenu des frames (option content). Si c'est le cas, la fonction analyse la frame (payload) et retourne true si le contenu est présent dans la frame, sinon false.

- **Void rule_matcher (Rule *rules_ds, ETHER_Frame *frame, int count)**

  Cette fonction est une des fonctions principales du projet. Elle a pour objectif, sur base des règles définies (rules_ds) de déterminer si la frame (frame) match avec une de ces règles. Dans un premier temps, elle analyse la frame pour en extraire les informations importantes dont nous aurons besoin. Dans un second temps, elle parcourt les règles une à une et pour chacune des règles, elle utilise les différentes fonctions de check pour dire s'il y a un match entre la frame et une des règles.

- **Int number_of_lines (FILE * file)**

  Cette fonction est utilisée pour calculer le nombre de règles présentes dans le fichier (file) en partant du principe que chaque ligne du fichier définit une seule règle.

- **void read_rules (FILE* file, Rule *rules_ds, int count)**

  Cette fonction est une des fonctions principales du projet. Elle a pour objectif, sur base du fichier où sont écrites les règles (file) de remplir une structure rule pour chacune des règles (rules_ds). Cette fonction lit donc ligne par ligne le fichier, et pour chaque ligne, les éléments sont découpés pour ensuite utiliser ceux utiles dans la structure rule.

- **Void my_packet_handler (u_char *args,const struct pcap_pkthdr *header,const u_char *packet)**

  Cette fonction sera appelée à chaque fois qu'une frame sera réceptionnée. La frame sera dans un premier temps traitée avec la fonction populate_packet_ds

(définie dans le fichier populate.c) pour avoir une structure ETHER_Frame qui facilite la manipulation de la frame.
Une fois la structure représentant la frame créée, nous appliquons la fonction rule_matcher afin de déterminer s'il existe un match entre la frame et les règles définies. Il est important de préciser que l'argument args permet de passer à cette fonction la configuration utilisée, cf structure Configuration (l'ensemble des règles et leur nombre).

- **Int main (int argc, char *argv[])**

  Cette fonction est le point d'entrée du programme. C'est de là que nous faisons d'abord appel à la fonction read_rules et ensuite appel à la fonction pcap_loop. Cette dernière permet de rester en attente d'une frame sur une interface et de déclencher une fonction dans le cas où une frame est réceptionnée. La fonction déclenchée dans notre cas est my_packet_handler.

Populate.c

- **Int populate_packet_ds (const struct pcap_pkthdr *header, const u_char *packet, ETHER_Frame *custom_frame)**

  Cette fonction permet de traiter une frame brute (tel que reçue directement sur l'interface) pour obtenir une structure ETHER_Frame représentant les éléments importants de la frame de base et beaucoup plus simple à la manipulation, ce qui nous facilite grandement la tache.

Populate.h

- **Struct sniff_udp**

  Cette structure, similaire aux structures sniff pour tcp ou ip, est utilisée pour représenter les informations sur l'en-tête UDP de la frame, dans le cas où cette dernière utilise le protocole de transport UDP.