Rapport du projet
Développement

Rapport du projet "My Secure IDS"

Sénéchal Julien
Wallemme Maxime

Sécurité des systèmes
Hénallux
Deuxième Bloc, groupe C2
Année académique 2020-2021

01 Janvier 2021

1 Résumé des fonctionnalités

Code disponible sur notre GitHub : ici

- Menu d'aide (option -h)
- Arguments pour choisir sa carte réseau (obligatoire : SUDO ./MSIDS **ETH0**)
- Possibilité de définir le nombre de frames à écouter/analyser (option -l [nombre])
- Possibilité de cibler le fichier de règles (option -a [chemin d'accès])
- Possibilité d'afficher les détails (les plus pertinents) sur chaque frame avec numérotation des frames (option -d)
- Possibilité d'afficher le contenu de chaque frame HTTP (option -D - Non utilisable en même temps que -d, le programme vous le dira)
- Une fonction pour afficher le protocol a été implémentée (plus d'informations par la suite)
- 2 actions sont possibles grâce au fichier de règles ("save" & "alert")

2 Description du main.c

- 4 variables globales y sont définies
  - Le compteur de chaque frame
  - Un entier qui transmet l'information que l'option -d est activé
  - Un entier qui transmet l'information que l'option -D est activé
  - Un entier qui transmet l'information que l'option -p est activé

```
//GLOBAL
int count_frame = 1;
int display_all_frames = 0;
int display_http = 0;
int print_alert = 0;
//////////////////////
```

FIGURE 1 – Les variables globales

- La définition de la fonction de loopback (my_packet_handler)
  - On lui renseigne en plus des informations de base, une structure "Arguments" castée en unsigned char. Cette structure va nous permettre de faire parvenir a notre rule_matcher le pointeur de structure "Rule" et le nombre de règles.
  - On appelle la fonction "populate_packe_ds"
  - On affiche les éléments voulu en fonction de si l'option -d ou -D est activée
  - Appel la fonction rule_matcher
  - On fini par "clean le buffer", ce bout de code permet de ne pas avoir d'informations "parasite" a la prochaine frame si celle-ci ne correspond ni au protocole IP ni à l'ARP

```c
void my_packet_handler(u_char *args,const struct pcap_pkthdr *header,const u_char *packet){

	Arguments * args_lst = (Arguments *) args;
	ETHER_Frame frame;
	populate_packet_ds(header,packet,&frame,display_all_frames,count_frame);

	if(display_all_frames){
		if (frame.ethernet_type == ARP){
			printf("Protocol : ARP\n");
		}
		else if(frame.ethernet_type == IPV4){
			switch (show_protocol(&frame))
			{
			case 1:
				printf("Protocol : TCP\n");
				break;
			case 2:
				printf("Protocol : UDP\n");
				break;
			case 3:
				printf("Protocol : HTTP\n");
				break;
			case 4:
				printf("Protocol : HTTPS\n");
				break;
			case 6:
				printf("Protocol : FTP\n");
				break;
			case 0:
				printf("Protocol : Not referenced\n");
				break;
			}

			if(show_protocol(&frame) == 3 || show_protocol(&frame) == 6){
				printf("\n~~~~~DATA~~~~~\n");
				print_payload(frame.data.data.data_length,frame.data.data.data);
			}
		}

	}

	else if(display_http){

		if(show_protocol(&frame) == 3){

			print_payload(frame.data.data.data_length,frame.data.data.data);
		}
	}

	rule_matcher(args_lst->lst_rules, &frame, args_lst->n_rules, print_alert);

	count_frame++;

	//Clean buffer
	frame.ethernet_type = 0;
	frame.data.protocol_ip = 0;
}
```

FIGURE 2 – Fonction My_Packet_Handler

- La fonction main
  - On vérifie les arguments de la commande de lancement de notre programme et on modifie les variables liées en conséquence

```c
int is_interface = 0;
const char* device;
int nloop;
int is_nloop = 0;
int is_help = 0;
int is_address = 0;
char* file_address;

for(int i = 0; i < argc; i++){

	//Setting up the interface
	if(argv[1]!=NULL){

	    device = argv[1];
	    is_interface = 1;
	}

	//Setting up the number of loops
	if(!strcmp(argv[i], "-l")){
		if(argv[i + 1] == NULL){
			break;
		}
		else if(sscanf(argv[i + 1], "%d", &nloop) != 1){ //Check that the argument is an integer
			printf("\"%s\" is a bad argument for loop -l\n",argv[i + 1]);
			break;
		}
		char* endptr;
		nloop = (int)strtol(argv[i+1], &endptr, 10); //To convert a pointer of char in an integer
		is_nloop = 1;
	}

	//Display the help menu
	else if(!strcmp(argv[i],"-h")||!strcmp(argv[i],"--help")){
		print_help_menu();
		is_help = 1;
	}

	//Show all frames
	else if(!strcmp(argv[i],"-d")){
		display_all_frames = 1;
	}

	//Show all frames
	else if(!strcmp(argv[i],"-D")){
		display_http = 1;
	}

	//Show alerts
	else if(!strcmp(argv[i],"-p")){
		print_alert = 1;
	}

	//Set the rules file
	else if(!strcmp(argv[i],"-a")){
	    if(argv[i+1] == NULL){
		printf("Rules file argument missing\n");
		exit(1);
	    }
	    file_address = argv[i+1];
	    is_address = 1;

	}
}
```

FIGURE 3 – Traitement des arguments dans le main

— Si l'interface n'est pas renseignée, on arrête le programme avec un message d'erreur

```
else if(!is_help){
        printf("Missing arguments. Do \"ids --help\" or \"ids -h\" for more information.\n");
}
```

Figure 4 – Erreur en cas de manque d'interface

— Si l'option -h est actif, alors quelques soit les autres arguments, on print le menu d'aide grâce à la fonction "print_help_menu()"
— On ouvre le fichier de règles (par la même occasion on vérifie qu'on arrive a le lire), et on compte le nombre de lignes que l'on a (c'est à dire le nombre de règles)

```
//Checking if an address was entered
if(!is_address){
    file_address = "ids.rules";
}

//Check the number of rules
FILE *file = fopen(file_address,"r");
int n_rules = 0;
char rule[MAXLINE];

if(file == NULL){
        printf("An error occurred while reading the rules file\n");
        exit(1);
}
while(fgets(rule,MAXLINE,file)!= NULL){
        n_rules ++;
}
printf("%d rules have been found in %s\n",n_rules,file_address);
fclose(file);
```

Figure 5 – Lecture du nombre de règles

— On lance la fonction "read_rules()"

```
//Read all rules
FILE *rule_file = fopen(file_address,"r");
Rule lst_rules[n_rules];
read_rules(rule_file, lst_rules, n_rules);
fclose(rule_file);
```

Figure 6 – Read rules

— Si l'option -l n'a pas été ajoutée, alors on défini le nombre de frames à lire à 1000

```
if(!is_nloop){
        nloop = 1000;
}
```

Figure 7 – Nombre de frames

— On crée la structure "Arguments" qui contient le pointeur de structure "Rule" ainsi que le nombre de règles

```
Arguments argument_loop;
for(int i = 0; i < n_rules; i++){
        argument_loop.lst_rules[i].action = lst_rules[i].action;
        argument_loop.lst_rules[i].protocol = lst_rules[i].protocol;
        argument_loop.lst_rules[i].port_dst = lst_rules[i].port_dst;
        argument_loop.lst_rules[i].port_src = lst_rules[i].port_src;

        strcpy(argument_loop.lst_rules[i].ip_dst, lst_rules[i].ip_dst);
        strcpy(argument_loop.lst_rules[i].ip_src, lst_rules[i].ip_src);
        strcpy(argument_loop.lst_rules[i].options, lst_rules[i].options);
}
argument_loop.n_rules = n_rules;
```

FIGURE 8 – Création de la structure Arguments

— On lance l'écoute de l'interface

```
pcap_loop(handle, nloop, my_packet_handler, (u_char*)&argument_loop);
```

FIGURE 9 – Lecture sur l'interface

3 Implémentation de la fonction "read_rules"

- Se trouve dans le fichier "rules.c"
- Fonctionnement
  1. Il reçoit un fichier sous forme d'une structure "FILE", un pointeur de structure "Rule" ainsi que le nombre de structure "Rule" que renvoie le pointeur
  2. Grâce a une boucle for, nous allons boucler sur chaque ligne du fichier afin d'en retirer les règles et de remplir nos structures "Rule". Une règle = une structure
  3. Le programme défini chaque élément de la ligne grâce à la fonction "strtok()"
  4. Pour des raisons de facilité pour le rule_matcher, les variables "action" et "protocol" sont défini par des entiers
  5. Action :
     — 1 = "alert"
     — 2 = "save"
  6. Protocol :
     — 0 = Not implemented
     — 1 = TCP
     — 2 = UDP
     — 3 = HTTP
     — 4 = HTTPS
     — 5 = ARP
     — 6 = FTP
  7. Enfin, la fonction termine chaque ligne en ajoutant les options sous forme de chaîne de caractères. Par soucis de lisibilité, une boucle va permettre d'en enlever la parenthèse qui commence cette chaîne de caractères.

4 Implémentation de la fonction "rule_matcher"

- Se trouve dans le fichier "rules.c"
- Reçoit le pointeur de structure "Rule" pré-remplit par read_rules, le pointeur de la structure "ETHER_frame", le nombre de règles, et un entier définissant si l'option "-p" fut activée ou non
- Arborescence de if et de else :
  1. Vérifie que le protocole corresponde

2. On fait le tri entre les divers protocoles car la suite de l'arborescence ne sera pas la même pour tous (1er groupe : TCP, HTTP, HTTPS, FTP - 2ème groupe : UDP - 3ème groupe : ARP)
3. Vérification que l'addresse IP source corresponde
4. Vérification que le port source corresponde
5. Vérification que l'addresse IP de destination corresponde
6. Vérification que le port de destination corresponde
7. Vérification de l'action

- Action "alert"
  1. On vérifie qu'il ne s'agisse pas de l'HTTPS, sinon voir point 6
  2. On vérifie que l'option "content" se trouve dans les options grâce à la fonction "strstr()", si pas voir point 5
  3. Etant donné qu'on utilise "strtok()" (fonction destructive) pour récupérer le contenu, on travaille sur une sauvegarde des options
  4. On print le message d'alerte si l'option a été activée et on ajoute l'alerte au syslog - **Fin de l'option content**
  5. On print le message d'alerte si l'option a été activée et on ajoute l'alerte au syslog - **Fin de l'alerte**
  6. On print le message d'alerte si l'option a été activée et on ajoute l'alerte au syslog - **Fin de l'alerte si https**
- Action "save"
  1. On s'assure que le fichier dans lequel on va écrire les informations se trouve bien dans les options, si pas, on en crée un au nom de "save_msids"
  2. On inscrit les informations utiles dans le fichier pour chaque frame répondant a cette règle, les informations dépendent du protocole utilisé. Les informations minimale sont le protocole et l'heure.

5 Modifications apportée au populate.c

- La fonction populate_packet_ds() reçoit en plus la variable contenant le numéro de la frame et la variable qui défini si les informations de la trames doivent être print (option -d)
- Ajout de la prise en charge de l'UDP
- On garde en mémoire (dans la structure custom_ip) le protocole sous forme d'entier pour être repris lors de la fonction "show_protocol()"

6 Fichier protocol.c

- Fonction show_protocol()
- Cette fonction va simplement retourner un entier qui correspond au protocole de couche la plus élevé possible (selon ce qui a été implémenté).
  1. TCP
  2. UDP
  3. HTTP
  4. HTTPS
  5. ARP
  6. FTP
- HTTP
  - On va simplement vérifier que les 4 premiers caractères du payload soient "GET " ou "HTTP" (GET pour la requête et HTTP pour la réponse).
- HTTPS
  - Contraiement à l'http, on vérifie que GET et HTTP soient bien illisible et on vérifie que le port utilisé est bien le port 443
  - Il était pensé utiliser les requêtes TLS mais malheureusement le timing était trop serré
- FTP
  - Pour savoir si c'est du FTP, on vérifie d'abord que ça soit un segment TCP, ensuite si un des ports (destination ou source) est le 20 ou le 21, on retourne 6, ce qui équivaut au FTP

7 Fichier help.c

— Contient une simple fonction qui print le menu d'aide

8 Modification de populate.h

- Ajout des différentes librairies dont on a eu besoin
- Ajout des différents fichiers du code
- Ajout des #define u_char u_short et u_int parce que mon IDE était tout rouge à cause de toutes les erreurs qu'il m'affichait
- Ajout des structures pour l'UDP (custom_udp et sniff_udp)

```
struct sniff_udp{
        u_short port_src;       // Source port
        u_short port_dst;      // Destination port
        u_short len;            // Datagram length
        u_short crc;            // Checksum
};

#define SIZE_UDP 8
struct custom_udp
{
        int source_port;
        int destination_port;
        unsigned char *data;

} typedef UDP_Packet;
```

FIGURE 10 – Ajout des structures udp

- Ajout de la structure "Rule" et de la structure "Arguments"

```
struct ids_rule{
        int action;
        int protocol;
        char ip_src[IP_ADDR_LEN_STR];
        int port_src;
        char ip_dst[IP_ADDR_LEN_STR];
        int port_dst;
        char options[MAXLINE];
 } typedef Rule;

struct args_loop{
        Rule lst_rules[255];
        int n_rules;
}typedef Arguments;
```

FIGURE 11 – Ajout des structures

- Ajout de la variable "protocol" a la structure custom_ip
- Ajout des différents prototype de fonction (show_protocol, rule_matcher, read_rule)

```
int populate_packet_ds(const struct pcap_pkthdr *header, const u_char *packet,ETHER_Frame * frame, int dislay_all_frames, int count_frame);
void print_payload(int payload_length, unsigned char *payload);
int show_protocol(ETHER_Frame *frame);
void rule_matcher(Rule *rules_ds, ETHER_Frame *frame, int count, int print_alert);
void read_rules(FILE * file, Rule *rules_ds, int count);
```

FIGURE 12 – Ajout des prototypes

9 Quelques précisions

- L'ARP n'est pas vraiment implémenté, en fait le code ne récupère que le type ethernet. C'est pour ça que lorsqu'on crée une règle ARP, il faut absolument que les IP et les ports restent en "any"
- Le bout de code dans "rules_matcher" où l'on prend en charge la librairie "time.h" ne vient pas de nous (https ://www.developpez.net/forums/d558432/general-developpement/programmation-systeme/linux/heure-systeme-c/)
- Voici d'où j'ai tiré les structures pour l'UDP : http ://yuba.stanford.edu/ casado/pcap/section4.html
- L'archive fournie vient directement de notre GitHub : https ://github.com/Teckinfor/MySecureIDS
- Voici la commande de compilation utilisée : "gcc main.c populate.c rules.c protocol.c -o msids -lpcap -Wall"