

# Rapport projet développement : IDS

## Introduction :

Le projet suivant avait pour but de créer un IDS (Intrusion Detection System) en langage C.

Concept : L'IDS doit donc récupérer les paquets transitant sur une interface donnée et comparer ceux-ci avec une liste de règles données dans un fichier. Si le paquet enfreint une des règles, le logiciel doit faire remonter un message dans Syslog.

## Démarche :

Notre IDS va devoir donc : ouvrir une écoute sur une interface, boucler sur l'écoute, récupérer le paquet puis le décomposer pour garnir une structure qui pour être utilisée pour une comparaison. Nous devons également ouvrir le fichier de règles passé en argument et décomposer celui-ci dans un tableau de structure, chaque indice contenant une règle.

Enfin, une fonction comparera le paquet recueilli et décomposé dans une structure avec le tableau de structure contenant les termes de chaque règle.

## Aides et supports :

Nous avons utilisé la librairie :

PCAP.H : Permet de capturer un trafic réseau.

STRING.H : Manipulations des chaînes de caractères.

STDLIB.H : Manipulation et conversion des chaînes de caractères en valeurs numériques.

SYSLOG.H : Gestion de syslog.

STDIO.H : Gestion des entrées et sorties.

De plus, une librairie et du code déjà en place nous ont été fournis, celui-ci nous permet de garnir une structure à partir de la trame brute, fournie par la librairie PCAP.H

Développement :

```
//preparation de read_rules
FILE* file = fopen(argv[1], "r");

char c;
int count = 1;
while ((c=fgetc(file)) != EOF)
{
    if(c == '\n')
    {
        count++;
    }
}

fclose(file);
```

Nous récupérons le nom du fichier donné en argument et nous ouvrons son contenu dans une structure de type FILE, nous comptons chaque ligne grâce au caractère de retour à la ligne jusqu'au End Of File.

---

```
struct options_rule
{
    char type1[10];
    char msg1[50];
    char type2[10];
    char msg2[50];
} typedef Options;

struct ids_rule
{
    char type_rule[10];
    char protocol[10];
    char address_source[IP_ADDR_LEN_STR];
    int port_source;
    char direction[3];
    char address_destination[IP_ADDR_LEN_STR];
    int port_destination;
    Options idso;
    int totalRules;
} typedef Rule;
```

Le type de structure ids\_rules sera utilisé pour ranger chaque terme d'une règle. Une autre structure y est appelée, utilisée pour les options de règle.

---

```

Rule* rules_ds = (Rule*) calloc(count, sizeof(Rule));

rules_ds[0].totalRules = count;

file = fopen(argv[1], "r");

//printf("pre_read_rules\n");

read_rules(file, rules_ds);

fclose(file);

```

Une structure « rules\_ds » est créée, de l'espace est réservé pour elle en stack et elle est initialisé à 0. Le nombre de lignes comptées est attribué à un champ de cette structure. On fait appel à la fonction read\_rules() avec la structure file et rules\_ds en paramètres. On ferme la lecture du fichier.

```

void read_rules(FILE* file, Rule* rules_ds)
{
    //credits : cours
    char line[200];
    int ind = 0;
    while(fgets(line, 200, file) != NULL)
    {
        char tmp_line[strlen(line+1)];
        strcpy(tmp_line, line);
        char token[250];
        strcpy(token, strtok(tmp_line, " "));

        strcpy(rules_ds[ind].type_rule, token);

        strcpy(token, strtok(NULL, " "));
        strcpy(rules_ds[ind].protocol, token);

        strcpy(token, strtok(NULL, " "));
        strcpy(rules_ds[ind].address_source, token);

        strcpy(token, strtok(NULL, " "));
        rules_ds[ind].port_source = atoi(token);

        strcpy(token, strtok(NULL, " "));
        strcpy(rules_ds[ind].direction, token);

        strcpy(token, strtok(NULL, " "));
        strcpy(rules_ds[ind].address_destination, token);

        strcpy(token, strtok(NULL, " "));
        rules_ds[ind].port_destination = atoi(token);

        char* opt = strcpy(token, strtok(NULL, "("));

        strcpy(rules_ds[ind].idso.type1, strtok(opt, ":"));
        strcpy(opt, strtok(NULL, "\\");
        strcpy(rules_ds[ind].idso.msg1, opt);
    }
}

```

Dans `read_rules()` nous allons pouvoir, décomposer ligne par ligne le fichier `ids.rules`. Pour ce faire nous utiliserons la fonction `strtok()`, une chaîne de caractère lui est passé en paramètres avec un séparateur (dans ce cas des espaces). Le retour peut être stocké dans le champ de la structure voulu. La fonction `atoi()` permet de convertir une chaîne de caractère en entier, elle est utilisée pour les ports. Tant que la ligne ne sera pas nulle, la boucle décomposera chaque règle.

---

```
if(strcmp(opt, "") != 0 && strstr(opt, "content") != NULL)
{
    strcpy(rules_ds[ind].idso.type2, strtok(opt, ":"));
    strcpy(rules_ds[ind].idso.msg2, strtok(NULL, "\\;"));

    //printf("%s\n", rules_ds[ind].idso.msg2);
}
else
{
    strcpy(rules_ds[ind].idso.type2, "NULL");
    strcpy(rules_ds[ind].idso.msg2, "NULL");
    //printf("%s\n", rules_ds[ind].idso.msg2);
}
```

Cette partie de la décomposition servira pour la vérification sure « content ». On vérifie que le résultat de la décomposition précédente n'est pas la fin de la règle ET que le qu'il contient bien le terme « content ». Si c'est le cas, les deux derniers termes sont rangés dans leur champ. Sinon la chaîne « NULL » y sera inscrite.

---

```
//ecoute de l'interface eht0 ; credits : devdungeon
char* device = "eth0";
char error_buffer[PCAP_ERRBUF_SIZE];
pcap_t* handle;

handle = pcap_create(device,error_buffer);
pcap_set_timeout(handle, 10);
pcap_activate(handle);

int total_packet_count = 0;
```

Après avoir exécuté la fonction `read_rules()` et à l'aide des outils de PCAP.H, la fonction `pcap_create()` et `pcap_activate()` nous permettrons de créer une écoute sur l'interface renseignée dans la variable « device ». La variable « total\_packet\_count » permettra de renseigner le nombre de paquet à écouter (0 = infini).

---

```
//loop de l'ecoute des paquets
pcap_loop(handle, total_packet_count, my_packet_handler, (u_char*) rules_ds);
```

L'appel à la fonction `pcap_loop()` avec en paramètres : « handle » structure contenant les informations d'écoute ; « total\_packet\_count » ; la fonction `my_packet_handler()` devant encore être définie ; et un argument, dans notre cas, l'adresse de notre structure `rules_ds`. `Pcap_loop()` capture les paquets.

```
void my_packet_handler(u_char* args, const struct pcap_pkthdr* header, const u_char* packet)
{
    ETHER_Frame* custom_frame = (ETHER_Frame*) calloc(1, sizeof(ETHER_Frame));

    //Remplissage Struct Frame
    populate_packet_ds(header, packet, custom_frame);

    //Appel au Matcher
    rule_matcher((Rule*) args, custom_frame);

    free(custom_frame);
}
```

`My_packet_handler()` passée à `pcap_loop()` y est exécutée. Elle aura reçu les résultats de la capture qui seront eux-même passés en argument de la fonction `populate_packet_ds()` où ils y seront découpés et rangés dans la structure de type `ETHER_Frame`. La structure type `ETHER_Frame` rassemble tous les champs composant l'identification d'un paquet. Elle est initialisée en amont. Puis en toute fin l'espace mémoire est libéré.

Une fois une structure `ETHER_Frame` garnie, on fait un appel à `rule_matcher()` qui prend l'adresse fournie plus tôt de `rules_ds`, que l'on doit caster dans son type de structure, celui-ci ayant dû être changé pour pouvoir être passé à `pcap_loop()`. On lui donne aussi l'adresse de la structure contenant les informations du paquet.

```
void rule_matcher(Rule* rules_ds, ETHER_Frame* frame)
{
    for(int i = 0; i < rules_ds[0].totalRules; i++)
    {
        if(strcmp(rules_ds[i].address_source, "any") == 0 || strcmp(rules_ds[i].address_source, frame->data.source_ip) == 0)
        {
            if(rules_ds[i].port_source == 0 || rules_ds[i].port_source == frame->data.source_port)
            {
                if(strcmp(rules_ds[i].direction, "->") == 0)
                {
                    //printf("rule:%s frame:%s\n", rules_ds[i].address_destination, frame->data.destination_ip);
                    if(strcmp(rules_ds[i].address_destination, "any") == 0 || strcmp(rules_ds[i].address_destination, frame->data.destination_ip) == 0)
                    {
                        if(rules_ds[i].port_destination == 0 || rules_ds[i].port_destination == frame->data.destination_port)
                        {
                            //printf("%s %s\n", rules_ds[i].protocol, frame->protocol);
                            if(strcmp(rules_ds[i].protocol, frame->protocol) == 0)
                            {
                                if(strcmp(rules_ds[i].idso.type2, "NULL") == 0)
                                {
                                    printf("Le paquet enfreint une regle\n");

                                    if(strcmp(rules_ds[i].type_rule, "alert") == 0)
                                    {
                                        openlog("IDS", LOG_PID | LOG_CONS, LOG_USER);
                                        syslog(LOG_INFO, rules_ds[i].idso.msg1);
                                        closelog();
                                    }
                                }
                                else if(strstr((char*)frame->data.data, rules_ds[i].idso.type2) != 0)
                                {
                                    printf("Le paquet enfreint une regle\n");

                                    if(strcmp(rules_ds[i].type_rule, "alert") == 0)
                                    {
                                        openlog("IDS", LOG_PID | LOG_CONS, LOG_USER);
                                        syslog(LOG_INFO, rules_ds[i].idso.msg1);
                                        closelog();
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

`Rules_matcher()` va comparer chaque champ de `rules_ds` et de `frame`. Si les informations sont concordantes, le cas d'une vérification de « content » est fait et se termine par l'écriture du message de la règle dans le syslog.

### Aménagements :

Il aura fallu également intégrer un nouveau champ dans `ETHER_Frame` pour y ajouter un protocole, vérifier lors de `populate_packet_ds()` ainsi que l'ajout des informations de paquet dans le cas du protocole UDP.

```
struct custom_ethernet
{
    char source_mac[ETHER_ADDR_LEN_STR];
    char destination_mac[ETHER_ADDR_LEN_STR];
    int ethernet_type;
    int frame_size;
    char protocol[5]; //ajout
    IP_Packet data;
} typedef ETHER_Frame;
```

```
if((int)ip→ip_p==UDP_PROTOCOL)
{
    strcpy(custom_frame→protocol, "udp"); //ajout
    printf("\nUDP Handling\n");

    tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip); //ajout
    TCP_Segment custom_segment; //ajout

    custom_segment.source_port = ntohs(tcp→th_sport); //ajout
    custom_segment.destination_port = ntohs(tcp→th_dport); //ajout
    custom_packet.data = custom_segment; //ajout
    custom_frame→data = custom_packet; //ajout
}
```

```
if(custom_segment.source_port == 80)
{
    strcpy(custom_frame→protocol, "http"); //ajout
    //print_payload(payload_length, payload); //ajout
}
```

## Résultat :

The screenshot shows a Kali Linux virtual machine with three terminal windows. The top window shows the compilation of the IDS rules using `sudo gcc -Wall -o ids_main.c populate.c -lpcap` and `sudo ./ids ids.rules`. The middle window shows a netcat listener on `172.16.0.1 9999` receiving a connection from `172.16.0.1`. The bottom window shows a log of network traffic, including a UDP packet from `172.16.0.1` to `172.16.0.1` on port `9999`, which is blocked by the IDS rule `127`.

```
simon@kali: ~/Documents/ids_v1.0
File Actions Edit View Help
sudo: a password is required

(simon@kali)-[~/Documents/ids_v1.0]
$ sudo gcc -Wall -o ids_main.c populate.c -lpcap
[sudo] password for simon:

(simon@kali)-[~/Documents/ids_v1.0]
$ sudo ./ids ids.rules

IPV4 packet: 2048
UDP Handling
Le paquet enfreint une règle
IPV4 packet: 2048
ARP packet: 2054
ARP packet: 2054

simon@kali: ~
File Actions Edit View Help
$ nc -u 172.16.0.1 9999
dsfs
^C

(simon@kali)-[~]
$ nc -u 172.16.0.1 9999
sdf

simon@kali: ~
File Actions Edit View Help
Jan 3 14:37:17 kali kernel: [ 1333.505887] No guest source window
Jan 3 14:37:17 kali kernel: [ 1333.511003] No guest source window
Jan 3 14:37:17 kali kernel: [ 1333.516144] No guest source window
Jan 3 14:37:17 kali kernel: [ 1333.516935] No guest source window
Jan 3 14:37:17 kali kernel: [ 1333.524768] No guest source window
Jan 3 14:37:17 kali kernel: [ 1333.528266] No guest source window
Jan 3 14:37:17 kali kernel: [ 1333.532630] No guest source window
Jan 3 14:37:17 kali kernel: [ 1333.539034] No guest source window
Jan 3 14:37:17 kali kernel: [ 1333.550580] No guest source window
Jan 3 14:37:17 kali kernel: [ 1333.554742] No guest source window
Jan 3 14:37:17 kali kernel: [ 1333.562079] No guest source window
Jan 3 14:37:17 kali kernel: [ 1333.572700] No guest source window
Jan 3 14:37:17 kali kernel: [ 1333.582831] No guest source window
Jan 3 14:37:17 kali kernel: [ 1333.591907] No guest source window
Jan 3 14:37:17 kali kernel: [ 1333.599809] No guest source window
Jan 3 14:37:17 kali kernel: [ 1333.613786] No guest source window
Jan 3 14:37:17 kali kernel: [ 1333.634700] No guest source window
Jan 3 14:39:01 kali CRON[1600]: (root) CMD ( [ -x /usr/lib/php/sessioncle
an ] && if [ ! -d /run/systemd/system ]; then /usr/lib/php/sessionclean; fi
)
Jan 3 14:39:32 kali systemd[1]: Starting Clean php session files ...
Jan 3 14:39:32 kali systemd[1]: phpsessionclean.service: Succeeded.
Jan 3 14:39:32 kali systemd[1]: Finished Clean php session files.
Jan 3 14:40:56 kali IDS[1544]: UDP traffic bind port is forbidden

(simon@kali)-[~]
$
```

Exemple d'une connexion UDP ayant été reconnue comme enfrenant une règle d'ids.rules.



## Sources :

<https://www.devdungeon.com/content/using-libpcap-c>

<https://stackoverflow.com/questions/24768543/how-to-pass-multiple-arguments-to-pcap-loop-pcap-handler>

Cours développement

## Git :

<https://github.com/Simoaw/IDS>