

WEB FILTERING con SNORT REACT

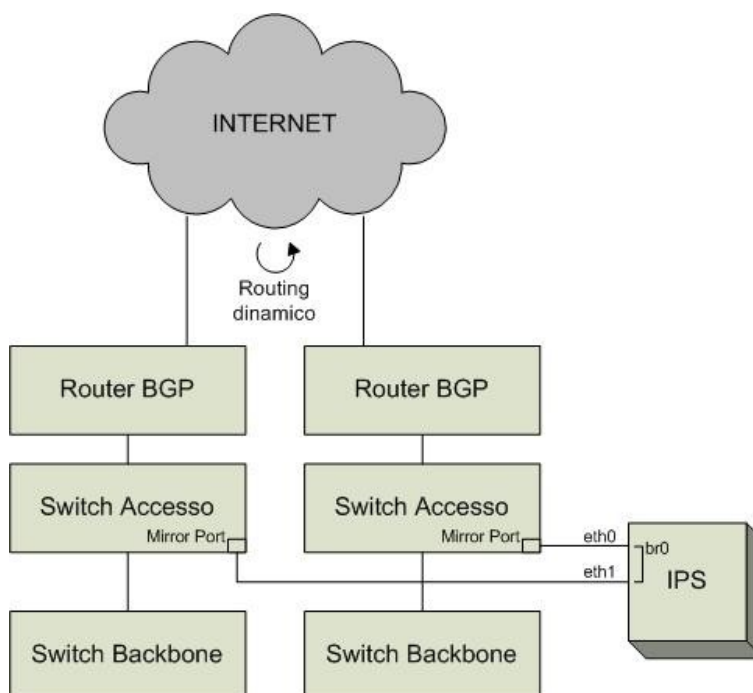
Pierpaolo Palazzoli, Brescia, Italy

Fabio Mostarda, Brescia, Italy

Claudia Ghelfi, Brescia, Italy

CONTESTO

Si supponga di agire in un contesto simile al seguente. Entrambi gli “Switch di accesso” sono dotati di una mirror port su cui replicano il traffico che li attraversa. Tale traffico viene poi analizzato da un’IPS realizzato con Snort in modalità Flexible Response.



Si supponga poi di voler operare “web filtering”. E’ allora possibile, grazie a Snort, segnalare particolari siti visitati piuttosto che dropparne le connessioni ed impedirne quindi l’accesso, oppure, redirezionare l’utente verso una particolare pagina html che spieghi il perché della limitazione.

Quest’ultima possibilità è offerta da una particolare modalità operativa di Snort detta “react”. Il “react” è subordinato ad un ovvio match con un elenco di regole.

Tali regole sono strutturate in modo simile a quelle che operano il reset; se ne riporta una a titolo esemplificativo:

```

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:
"SNORTATTACK AAMS ITALIAN LAW"; flow: to_server,established;
content:"it.888.com"; reference:url,www.aams.it; classtype: policy-
violation;sid:5002020; react: block, msg ; )

```

Questa regola blocca la sessione (*react: block*) e redireziona l'utente verso una pagina html, definita nel file "sp_react.c", in cui appare il messaggio impostato nella regola (*msg*)

Osservazioni

- ➔ Solo Snort propone il funzionamento in modalità Flexible Response, tutti i dispositivi analoghi infatti operano unicamente Inline.
- ➔ Snort offre due modi alternativi di operare in Flexible Response: FlexRespo e FlexRespo2. Il "react" è presente solo per FlexRespo.
- ➔ È da considerare che Snort flexresp è un dispositivo passivo e quindi la sua efficacia è direttamente dipendente dall'occupazione della CPU e dai collegamenti del sistema su cui è implementato, dalla memoria disponibile, dalla situazione di I/O e dalle latenze di rete.
- ➔ A causa del protocollo di routing dinamico, è necessario, per riassemblare le sessioni, creare un bridge virtuale tra le interfacce eth0 ed eth1 su cui analizzare il traffico.

REALIZZAZIONE

La realizzazione di un IPS che operi il "react" segue questi passi:

1. Installazione dei pacchetti propedeutici:

```

apt-get install vim gcc make build-essential libtool automake autoconf flex
bison libpcrc3-dev psmisc ethtool apache2 ntop ntpdate subversion apache2
kernel-package libncurses5-dev fakeroot wget bzip2 flex bison zlibc zlib1g-dev
libnet0 libnet0-dev

```

2. Creazione del bridge br0:

Si inserisce il testo seguente nel file /etc/rc.local

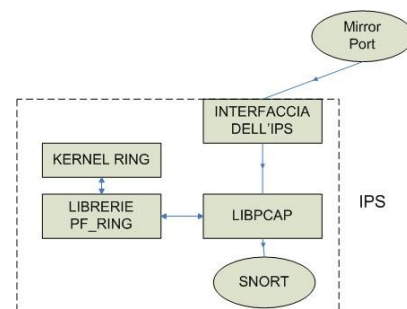
```

ifconfig eth0 0.0.0.0 promisc up
ifconfig eth1 0.0.0.0 promisc up
brctl addbr br0
brctl addif br0 eth0
brctl addif br0 eth1
ifconfig br0 up

```

3. Implementazione del PF_RING e installazione delle librerie Libpcap relative:

PF_RING è una tipologia di socket di rete che accresce notevolmente la velocità di cattura dei pacchetti.



Per questo motivo, se inserita nella struttura dell'IPS, come qui schematizzato, ne aumenta le prestazioni.

Innanzitutto si scarica il PF_RING, in questo caso l'ultima versione:

```
cd /usr/src
svn co https://svn.ntop.org/svn/ntop/trunk/PF_RING/ scarico ultima versione
pf_ring
```

Si verifica il kernel attualmente in uso:

```
uname -a
```

Si modifica lo script scaricato affinché generi la **patch** del kernel che si preferisce, in questo caso si è scelto di creare la patch per quello in uso:

```
cd /usr/src/PF_RING/
vi mkpatch.sh
```

edito kernel giusto

```
VERSION=${VERSION:-2}
PATCHLEVEL=${PATCHLEVEL:-6}
SUBLEVEL=${SUBLEVEL:-26}
```

Si lancia lo script che genera la patch:

```
sh ./mkpatch.sh
cd workspace/
cd linux-2.6.26-1-686-smp-PF_RING
cp /boot/config-2.6.26-1-686-bigmem .config
```

Si abilita il pfring cosicché ne venga tenuto conto nella generazione del **nuovo kernel**:

```
make menuconfig
```

Si seleziona *Networking -> Networking Options* e ci si assicura che PF_RING socket sia abilitato.

Si esce dal menù salvando le modifiche effettuate.

Ora è possibile generare il pacchetto contenente il nuovo kernel e installarlo:

```
make-kpkg clean
fakeroot make-kpkg -initrd -revision=pfring.1.0 linux-image
cd ../
dpkg -i linux-image-2.6.26_pfring.1.0_i386.deb
```

L'utilizzo di questo nuovo kernel di default ad ogni boot va impostato nel seguente file:

```
vi /boot/grub/menu.lst
```

Creiamo il file :

```
vi /etc/modprobe.d/options
```

inserendo la riga : options ring num_slots=32740 transparent_mode=0

Dove 65535 è la memoria occupata dal buffer ring. (scegliere a seconda della disponibilità, in caso sia maggiore bisogna cambiare nel grub la vmalloc=256M esempio di 256M)

Con il riavvio della macchina si verifica che il nuovo kernel venga effettivamente caricato:

```
reboot
```

Copio il sorgente del ring nella directory linux:

```
cp /usr/src/PF_RING/workspace/linux-2.6.26-1-686-smp-  
PF_RING/include/linux/ring.h /usr/include/linux/  
cd /usr/src/PF_RING/userland  
make
```

A questo punto si installano le librerie **Libpfring**:

```
cd lib  
gcc -shared -Wl,-soname -Wl,libpfring.so.0.9.7 -o libpfring.so.0.9.7 *.o -lc  
cp libpfring.a libpfring.so.0.9.7 /usr/local/lib  
cp pfring.h /usr/local/include  
ln -s /usr/local/lib/libpfring.so.0.9.7 /usr/local/lib/libpfring.so  
ldconfig  
ldconfig -v |grep pfring
```

Mediante quest'ultima istruzione si controlla che le librerie installate siano quelle corrette. La risposta deve quindi essere:

```
libpfring.so.0.9.7 -> libpfring.so.0.9.7
```

Infine, si installano le librerie **Libpcap-ring**:

```
cd /usr/src/PF_RING/userland  
wget http://www.tcpdump.org/release/libpcap-0.9.7.tar.gz  
tar -zxvf libpcap-0.9.7.tar.gz  
cd libpcap-0.9.7  
mv pcap-int.h pcap-int.h.orig  
mv pcap-linux.c pcap-linux.c.orig  
cp ../libpcap-0.9.7-ring/pcap* .  
./configure CPPFLAGS="-I/usr/local/include" LDFLAGS="-L/usr/local/lib"  
CFLAGS="-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE  
-D_FILE_OFFSET_BITS=64"  
make && gcc -shared -Wl,-soname -Wl,libpcap.so.`cat VERSION` -o  
libpcap.so.`cat VERSION` *.o -lc  
make install && cp libpcap.so.0.9.7 /usr/local/lib  
ldconfig -v |grep pcap
```

Mediante quest'ultima istruzione si controlla che le librerie installate siano quelle corrette. La risposta deve quindi essere:

```
libpcap.so.0.9.7 -> libpcap.so.0.9.7
```

4. Installazione di Snort:

Si installa Snort eseguendo:

```
wget http://www.snort.org/dl/snort-2.8.3.2.tar.gz  
tar zxvf snort-2.8.3.1.tar.gz
```



A questo punto è possibile aggiungere una patch che permetta di personalizzare la pagina html di redirezione e che consenta il funzionamento del “react” anche in ambiente Windows.

E’ possibile scaricare la **patch** realizzata dal team di Snortattack:

```
wget http://www.snortattack.org/sp\_react.c
```

per sistemarla poi nella directory appropriata:

```
cp sp_react.c snort-2.8.3.2/src/detection-plugins/
```

Se si desidera personalizzare il comportamento del react plugin vedere l'Appendice A

```
cd snort-2.8.3.1
```

```
./configure --enable-flexresp --enable-memory-cleanup --enable-linux-smp-  
stats --enable-pthread
```

```
vi src/Makefile
```

Modifico il Makefile in modo tale che:

si aggiunga la seguente riga

```
LDFLAGS = -L/usr/local/lib -lpfring -lpcap
```

ed esista questo testo:

```
CPPFLAGS = -I/usr/local/include
```

Se tutto fin qui è andato a buon fine, si esegue:

```
make
```

```
make install
```

```
cd ..
```

```
mkdir /var/log/snort
```

5. Definizione delle regole di Snort:

Le regole vengono inserite in /etc/snort/rules/.

Al termine di ogni regola deve essere inserito, come illustrato in precedenza: *react: block, msg* ; che si riferisce al “react”.

6. Avvio di Snort:

Infine si avvia Snort:

```
snort -A fast -i br0 -b -d -D -c /etc/snort/snort.conf -l /var/log/snort/law/  
--nolock-pidfile
```

Appendice A

esempio di sp_react.c , dalla riga 313 alla 316:

```
313> char tmp_head[] = "HTTP/1.1 302 FOUND\r\nLocation:
http://serverx:80\r\nServer: Snort/2.8.3.2\r\nConnection:
Close\r\nContent-Type: text/html\r\n\r\n";

314> char tmp_buf1[] = "<HTML><HEAD><TITLE>Snort</TITLE></HEAD><BODY
BGCOLOR=\"#FFFFFF\"><CENTER><BR><H1>Snort!</H1>Version ";

315> char tmp_buf2[] = " Snortattack patch! <H1><BR><BR><FONT
COLOR=\"#FF0000\">You are not authorized to open this site!
</FONT><BR><BR></H1><H2>";

316> char tmp_buf3[] = "<BR></H2><BR></BODY></HTML>";
```

ecco cosa modificare:

La riga 313 contiene l'header http di risposta, è suddivisa in varie parti delimitate dai caratteri speciali `\r\n` :

Prima parte:

- HTTP/1.1 302 FOUND, se volete ridirigere la connessione ad un altro server http.
- HTTP/1.1 200 FOUND se vi basta che il react risponda con una semplice pagina html

Seconda Parte (*solo per 302*):

- Location: <http://serverx/path:80>, inserite il server a cui ridirigere il traffico, può essere un vostro server web oppure anche google.com o qualunque altro sito. Può essere un nome dns oppure direttamente un indirizzo ip. Potete specificare anche la porta tcp. Potete specificare anche un path.

Terza parte:

- Server: SnortAttack/2.8.3.2, nome server web e relativa versione, suddivisi da / . Ovviamente inserite quello che volete.

Il resto della riga 313 **non è da modificare**.



www.snortattack.org

Le righe 314, 315 e 316 servono solo se usate il metodo HTTP/1.1 200 FOUND e non il 302. Questo è il sorgente della pagina html che verrà visualizzata dal browser che subisce il react. Perché 3 variabili? Perché viene fatta una concatenazione con due valori: 1) versione di snort e 2) MSGid della rule che lancia il react.

Quindi:

Pagina HTML generata = char tmp_buf1[] + SNORT_VER + char tmp_buf2[] + RULE_MSGID + char tmp_buf3[]

Modificate pure questi 3 buffer con il contenuto html che più vi aggrada, avendo l'accortezza di escapare tutti i caratteri che devono essere escapati, è pur sempre un sorgente C.

non modificate nient'altro nel file, a meno che non siate programmatori C esperti si intende ;-)