CURSO DETECCION DE VULNERABILIDADES E INTRUSIONES

(Autor: Alejandro Corletti Estrada)

INTRODUCCION

En este texto, se tratará de desarrollar una metodología de trabajo que permita evaluar permanentemente el nivel de seguridad en el que se encuentra un sistema.

Para lograr este objetivo, se tratarán tres elementos fundamentales:

- Detector de Intrusiones (Se empleará la herramienta **Snort**)
- Detector de Vulnerabilidades (Se empleará la herramienta **Nessus**)
- Matriz de estado de seguridad (**MES**, desarrollada por el autor de este trabajo).

Las dos primeras herramientas, se tratarán rápidamente, pues no es un curso de empleo de las mismas, pero se destacarán los aspectos fundamentales para poder emplearlas y lo que es más importante, es que se hará hincapié en el trabajo con las reglas de ambas, para poder tener la certeza que lo que uno detecta como vulnerabilidad, la otra está en capacidad de generar un evento al escuchar este patrón por la red, y en caso de no hacerlo, durante el curso se operará sobre las mismas para poder crear las **local.rules** necesarias para que pueda asociarse una herramienta con la otra y estar seguro que las vulnerabilidades existentes en el sistema, serán detectadas por los sensores correspondientes SÍ o SÍ.

En cuanto a la matriz de seguridad, es una metodología de trabajo que se propondrá durante el curso, y que se ofrece para su uso y mejora, pues cada sistema en particular puede y debe ser ajustado en sus parámetros y gustos del administrador, para obtener los resultados deseados, es decir que todos los valores de esta matriz pueden ser optimizados y seguramente lo harán así a medida que trabajen con ella.

PARTE A: IDS

1. QUE ES UN IDS (Intrusion Detection System):

Un IDS es básicamente un sniffer de red, que se fue optimizando, para poder seleccionar el tráfico deseado, y de esta forma, poder analizar exclusivamente lo que se configura, sin perder rendimiento, y que luego de ese análisis en base a los resultados que obtiene, permite generar las alarmas correspondientes.

La primera clasificación que se debe tener en cuenta es que los hay y de red (Network IDSs o **NIDS**) y los hay de host (Host IDSs o **HIDS**). A lo largo de este curso, se tratarán exclusivamente los NIDS, pues son el núcleo de este trabajo, pero conociendo el funcionamiento de estos, es muy fácil pasar a los HIDS.

Luego existen otros criterios más que permiten catalogar estas herramientas, pero no se va a entrar en estos detalles, pues como ya se mencionó, aquí se va a emplear **Snort**, que es claramente uno de los productos líderes de esta tecnología (a criterio del autor, es el mejor) y es preferible dedicar la atención a aprender brevemente el funcionamiento del mismo.

Otro concepto es el de **DIDS** (Distributed IDSs), que es la evolución natural del trabajo con NIDS en redes complejas, donde es necesario armar toda una infraestructura de sensores, con la correspondiente arquitectura de administración, comunicaciones y seguridad entre ellos

Por último cabe mencionar que está naciendo el concepto de **ADSs** (Anomaly Detection Systems) que es una nueva variante de todo lo que se tratará en este texto.

Como todo elemento de seguridad, los NIDS pueden ser vulnerados, engañados, "puenteados" y atacados. Existen muchas estrategias y publicaciones de cómo evadir NIDS, por lo tanto a lo largo del curso se irá tratando de remarcar estos conceptos, para tenerlos en cuenta.

Hoy se debe considerar como un elemento imprescindible de todo sistema, es más se aprecia que sin estos, "sería como montar, en pleno siglo XXI una operación militar defensiva de noche y sin visores nocturnos".

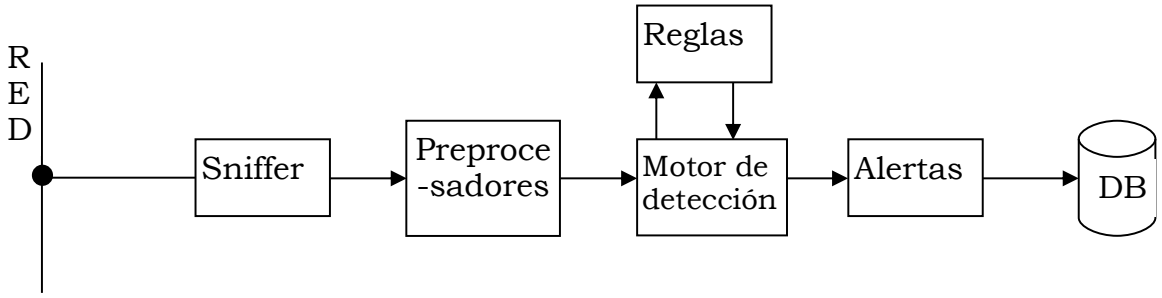
2. Breve descripción de Snort:

Esta herramienta fue creada por **Marty Roesch** en 1998. Nace simplemente como un Sniffer (o analizador de protocolos) al que luego se le fueron incorporando muchas otras opciones y en la actualidad cuenta con preprocesadores, plugins para bases de datos, varias opciones de envío de alarmas, diferentes esquemas de evaluar paquetes, conectores con Windows, consolas de administración gráficas, etc.

En concreto, Snort consiste de cuatro componentes básicos:

- El Sniffer.
- Los preprocesadores.
- El motor de detección.
- Las salidas.

El funcionamiento es el que se grafica a continuación:



Se detalla brevemente a continuación cada una de las partes:

a. El Sniffer

Es el paso inicial del funcionamiento de Snort, se relaciona directamente con la tarjeta de red, a la que coloca en modo "promiscuo", es decir que captura la totalidad del tráfico que circula por el cable, independientemente que vaya dirigido a su tarjeta de red o no. Con este primer paso se logra "escuchar" la totalidad de la información del sistema (se debe tener en cuenta el segmento en el que es colocado el dispositivo, pues si hubiere un switch de por medio, este dividiría los dominios de colisión y por lo tanto sólo se capturaría el tráfico correspondiente al segmento en el que se encuentre). El modo promiscuo se puede verificar con el comando "**ifconfig**", el cual indicará a través de la palabra *promisc*, si la tarjeta se encuentra en este modo

Para el funcionamiento del sniffer se apoya en la librería "**libpcap**", que es la misma que emplea el programa **tcpdump** (Snort hace uso de muchos programas ya existentes en el mundo GNU). Una

vez capturado cada paquete, se pasa al decodificador (esto lo realiza "***decode.c***")que es el responsable de interpretar la totalidad de los encabezados de cada nivel, desde enlace hasta aplicación.

b. Los preprocesadores

Los preprocesadores son un elemento fundamental para el rendimiento de Snort. Como su nombre lo indica, realizan un análisis previo de los paquetes capturados, confrontándolos con sus **plug-ins**, para evitar seguir escalando todo el volumen de información y poder realizar evaluaciones más simples. Se tratarán en detalle más adelante, pero en resumen sus tareas son la estandarización de formatos, decodificación, seguimiento de conexiones, análisis scan, etc.

c. El motor de detección

Esta parte es el corazón de Snort, toma la información que proviene de los preprocesadores y sus plug-ins y se verifica con el conjunto de reglas, si existe alguna correspondencia con estas últimas, envía una alerta. El tratamiento de las reglas se hará más adelante en este texto.

d. Las salidas

En la actualidad, Snort permite varios tipos de salidas al detectar una alerta. Pueden ser manejadas en forma local, a través de los logs, enviadas a otro equipo por medio de sockets de UNIX, SMB de Windows, protocolo SNMP, e-mails (SMTP), SMSs, formato XML, etc.

En cuanto al formato de las alertas es también muy variado y su almacenamiento en diferentes tipos de bases de datos (Se inició con **MySQL**, actualmente se aconseja **Postgres**).

La presentación visual de las mismas ofrece a su vez varias alternativas, y existen muchos plug-ins para Perl, PHP, y todo tipo de servidores Web (La aplicación más tradicional es ACID y en este texto se empleó la integración de la misma con la consola Snort Center).

3. ¿Cómo se usa Snort?

Si bien en muchos textos se suele clasificar el empleo de Snort, como Sniffer, Logger e IDS y se tratan por separado, aquí para simplificar el empleo y llegar a entender su uso más básico, se detallarán los comandos más comunes para emplear Snort sin estas subdivisiones.

-v coloca a snort en modo sniffer (Sólo encabezados de nivel transporte).

-d incluye los encabezados de nivel red.

-e incluye nivel de enlace.

EJEMPLO 1:

 # snort -dev

```
Running in packet dump mode
Log directory = /var/log/snort

Initializing Network Interface eth0

     --== Initializing Snort ==--
Initializing Output Plugins!
Decoding Ethernet on interface eth0

     --== Initialization Complete ==--

-*> Snort! <*-
Version 2.0.0 (Build 72)
By Martin Roesch (roesch@sourcefire.com, www.snort.org)
```

Se puede apreciar en el ejemplo anterior, un inicio de Snort con los mensajes de que todo ha sido realizado satisfactoriamente.

-l se emplea para indicar el directorio en el que se desean almacenar las alarmas.

-b almacena datos en formato binario (Igual que tcpdump), es muy útil para luego ser leídos con cualquier analizador de protocolos (como Ethereal).

-L Se debe emplear junto con la opción -b para indicar el nombre del archivo en el que se almacenarán los datos.

-r para leer cualquier archivo guardado en formato binario.

port (puede operar sólo o también con **src** y **dst**): Indica el puerto deseado.

host indica un host para detectar únicamente este rango (también se puede emplear **src** y **dst**).

net indica una red para detectar únicamente este rango (también se puede emplear **src** y **dst**).

NOTA: como en casi todos los comandos UNIX, la opción "**not**", niega la ejecución del mismo. Los operadores lógicos "**or**" y "**and**" también son válidos, por lo tanto puede ser empleados con la mayoría de las opciones de Snort.

Se debe tener en cuenta que Snort está fuertemente relacionado al lenguaje de comandos **BPF** (Berkeley Packet Filter), es decir la masa de las opciones empleadas en sistemas **UNIX**, este lenguaje permite especificar, hosts, redes, puertos, protocolos, etc. (la lista de estos comandos puede consultarse en www.tcpdump.org).

EJEMPLOS 2:

 # snort -dev -l /var/snort/log (Almacenará en formato log en ese path).

 # snort -dev host 10.1.1.1 (sólo operará con esa dirección IP)

 # snort -dev net 10.1.0.0/16 (Sólo operará con la red 10.1.x.x)

 # snort -b -L /var/archivo1 (Almacenará en formato binario en ese path).

 # snort -b -r /var/archivo1 (leerá ese archivo desde binario y en ese path).

 # snort -dev host 10.1.1.1 and port 80 (operará sólo con esa dirección IP y sólo el puerto origen o destino 80).

 # snort -dev host 10.1.1.1 and dst port 80 (operará sólo con esa dirección IP y sólo el puerto destino 80).

 # snort -dev not net 10.1.0.0/16 (ignorará la red 10.1.x.x)

El empleo más potente es cuando se llama al archivo de configuración de Snort (Aquí es donde algunos textos lo denominan uso como IDS), para esto se emplea la opción "-c" y se aclara a continuación el path en donde se encuentra el archivo "**snort.conf**".

EJEMPLO 3:

---

snort -dev -l /var/snort/logs net 10.1.1.0/24 –c /var/snort/snort.conf (operará como IDS sólo con esa red, guardará los logs en el path indicado y su configuración será la guardada en el archivo snort.conf).

Para poder hacer funcionar a Snort como IDS de forma eficiente es importante entonces, poder entender y dominar el archivo **"snort.conf"**, que es el que le marca todas las funciones que lo caracterizan como IDS. Este archivo queda configurado por defecto al instalar Snort, y con esta configuración básica ya puede emplearse la herramienta, pero durante este texto, se tratará de centrar

la atención sobre este archivo, lo que será el pilar fundamental de este trabajo.

La configuración de este archivo, consta de cuatro pasos:

- Paso 1: Configuración de variables.
- Paso 2: Configurar las librerías dinámicas
- Paso 3: Configuración de preprocesadores.
- Paso 4: Configuración de salidas.
- Paso 5: Configuración de reglas.

Cada uno de estos pasos se tratan a continuación.

4. Variables:

En esta sección es posible agrupar varios tipos de elementos bajo el concepto de variables, las cuáles podrán emplearse luego para cualquier otra opción de configuración.

La sintaxis de las variables es: var <nombre_de_variable> <valor>

Para ser estrictos, se pueden emplear dos tipos de variables:

- Variables estáticas: son las que detallan rangos de direcciones.
- Variables dinámicas: Son las que tienen como <valor>, el nombre de una o varias variables estáticas, precedidas por el signo "$".

A continuación se presenta esta sección tal cual es en el archivo "**snort.conf**" y luego se describe cada variable.

```
##################################################
# Step #1: Set the network variables:
#
# You must change the following variables to reflect
# your local network. The variable is currently
# setup for an RFC 1918 address space.
#
```

```
# You can specify it explicitly as:
#
var HOME_NET 10.64.130.6/32
#
# or use global variable $<interfacename>_ADDRESS
# which will be always initialized to IP address and
# netmask of the network interface which you run
# snort at.  Under Windows, this must be specified
# as $(<interfacename>_ADDRESS), such as:
# $(\Device\Packet_{12345678-90AB-CDEF-1234567890AB}_ADDRESS)
#
# var HOME_NET $eth0_ADDRESS
#
# You can specify lists of IP addresses for HOME_NET
# by separating the IPs with commas like this:
#
# var HOME_NET [10.1.1.0/24,192.168.1.0/24]
#
# MAKE SURE YOU DON'T PLACE ANY SPACES IN YOUR LIST!
#
# or you can specify the variable to be any IP address
# like this:

#var HOME_NET any

# Set up the external network addresses as well.
# A good start may be "any"

var EXTERNAL_NET any

# Configure your server lists.  This allows snort to only look for attacks
# to systems that have a service up.  Why look for HTTP attacks if you are
# not running a web server?  This allows quick filtering based on IP addresses
# These configurations MUST follow the same configuration scheme as defined
# above for $HOME_NET.

# List of DNS servers on your network
var DNS_SERVERS $HOME_NET

# List of SMTP servers on your network
var SMTP_SERVERS $HOME_NET

# List of web servers on your network
var HTTP_SERVERS $HOME_NET

# List of sql servers on your network
var SQL_SERVERS $HOME_NET

# List of telnet servers on your network
var TELNET_SERVERS $HOME_NET

# Configure your service ports.  This allows snort to look for attacks
# destined to a specific application only on the ports that application
```

```
# runs on.  For example, if you run a web server on port 8081, set your
# HTTP_PORTS variable like this:
#
# var HTTP_PORTS 8081
#
# Port lists must either be continuous [eg 80:8080], or a single port [eg 80].
# We will adding support for a real list of ports in the future.

# Ports you run web servers on
var HTTP_PORTS 80

# Ports you want to look for SHELLCODE on.
var SHELLCODE_PORTS !80

# Ports you do oracle attacks on
var ORACLE_PORTS 1521

# other variables
#
# AIM servers.  AOL has a habit of adding new AIM servers, so instead of (AOL Instant Messenger)
# modifying the signatures when they do, we add them to this list of
# servers.
var AIM_SERVERS
[64.12.24.0/24,64.12.25.0/24,64.12.26.14/24,64.12.28.0/24,64.12.29.0/24,64.12.161.0/24,64.12.163.0/24,205.18
8.5.0/24,205.188.9.0/24]

# Path to your rules files (this can be a relative path)
var RULE_PATH ../rules

# Configure the snort decoder:
# ============================
#
# Stop generic decode events:
#
# config disable_decode_alerts
#
# Stop Alerts on experimental TCP options
#
# config disable_tcpopt_experimental_alerts
#
# Stop Alerts on obsolete TCP options
#
# config disable_tcpopt_obsolete_alerts
#
# Stop Alerts on T/TCP alerts
#
# config disable_ttcp_alerts
#
# Stop Alerts on all other TCPOption type events:
#
# config disable_tcpopt_alerts
#
# Stop Alerts on invalid ip options
```

```
#
# config disable_ipopt_alerts


# Configure the detection engine
# ==============================
#
# Use a different pattern matcher in case you have a machine with very
# limited resources:
#
# config detection: search-method lowmem
#
##################################################
```

- var HOME_NET: Permite establecer el o los rangos de todas las direcciones de la propia red. Se emplea notación **CIDR**. Separadas por coma (sin espacios) y encerradas entre [ ], permite contener todas las redes que se necesiten. Si se desea incluir cualquier valor, existe el parámetro “any”

- var EXTERNAL_NET: premite aclarar rangos de redes que se interpretarán como externas (suele emplearse “any”, pero en algunas ocasiones puede ser un parámetro importante a asignar).

- Var DNS_SERVERS
- Var SMTP_SERVERS
- Var HTTP_SERVERS
- Var SQL_SERVERS
- Var TELNET_SERVERS

  Todos estos servidores es importante incluirlos si se poseen las direcciones de los mismos, pues como se verá más adelante, muchas de las reglas de Snort, operan sobre servicios que prestan estos servidores, y en el caso de no conocer sus direcciones, emplea el valor por defecto que es $HOME_NET obligando al motor a trabajar con todas las IPs pertenecientes a los mismos. En cambio si se acotaron los valores de estos servidores, cada una de esas reglas, entrarán en juego únicamente cuando la dirección IP del datagrama que se esté tratando, se corresponda con uno de ellos, caso contrario, directamente se descarta mejorando sensiblemente el rendimiento de Snort.

- Var HTTP_PORTS
- Var SHELLCODE_PORTS
- Var ORACLE_PORTS

  Como se puede apreciar las variables anteriores están referidas a puertos. La lógica es la misma que la de los servidores, es decir, si se emplean determinados puertos en la red, para qué se va a obligar a Snort a analizar todos los existentes, si se puede reducir su espacio de búsqueda. Con

estas variables, se acotan los servicios y puertos presentes en la red.

- Var AIM_SERVERS: Esto especifica la lista de los servidores de AOL, ya viene en la configuración por defecto y se aconseja dejarlo como está.
- Var RULE_PATH: Permite definir dónde Snort debe buscar las reglas. Puede ser un path absoluto o relativo.

La ante última parte de la sección de variables permite configurar diferentes decodificadores:

- Config disable_decode alerts: Detiene los eventos genericos de decodificadores.
- Config disble_tcpopt_experimental_alerts: Detiene alertas sobre opciones de TCP experimentales.
- Config disable_ttcp_alert: : Detiene alertas sobre T/TCP.
- Config disable_tcpopt_alerts: Detiene alertas sobre todas las opciones de TCP.
- Config disable_ipopt_alerts: Detiene alertas sobre ociones IP no válidas.

La última parte de esta sección permite configurar el motor de detección.

- config detection: search-method lowmem (emplea un motor de menores recursos, en caso de trabajar con un hardware de muy bajos recursos.

5. Preprocesadores:

Con anterioridad se definió la función de estos módulos, ahora se tratarán un poco más en detalle cada uno de ellos.

El primer concepto a comprender, es que cada uno de ellos se activará y configurará a través del archivo "snort.conf" (una vez analizados los preprocesadores y las reglas, se verá en detalle este archivo) en la sección correspondiente a preprocesadores, la misma se presenta a continuación (Se resalta en negrita cada uno de ellos):

```
##################################################
# Step #2: Configure preprocessors
```

```
#
# General configuration for preprocessors is of the form
# preprocessor <name_of_processor>: <configuration_options>

# frag2: IP defragmentation support
# ------------------------------
# This preprocessor performs IP defragmentation.  This plugin will also detect
# people launching fragmentation attacks (usually DoS) against hosts.  No
# arguments loads the default configuration of the preprocessor, which is a
# 60 second timeout and a 4MB fragment buffer.

# The following (comma delimited) options are available for frag2
#    timeout [seconds] - sets the number of [seconds] than an unfinished
#                        fragment will be kept around waiting for completion,
#                        if this time expires the fragment will be flushed
#    memcap [bytes] - limit frag2 memory usage to [number] bytes
#                     (default:  4194304)
#
#    min_ttl [number] - minimum ttl to accept
#
#    ttl_limit [number] - difference of ttl to accept without alerting
#                         will cause false positves with router flap
#
# Frag2 uses Generator ID 113 and uses the following SIDS
# for that GID:
#  SID     Event description
# -----   -------------------
#   1      Oversized fragment (reassembled frag > 64k bytes)
#   2      Teardrop-type attack

preprocessor frag2

# stream4: stateful inspection/stream reassembly for Snort
#--------------------------------------------------------------------------
# Use in concert with the -z [all|est] command line switch to defeat
# stick/snot against TCP rules.  Also performs full TCP stream
# reassembly, stateful inspection of TCP streams, etc.  Can statefully
# detect various portscan types, fingerprinting, ECN, etc.

# stateful inspection directive
# no arguments loads the defaults (timeout 30, memcap 8388608)
# options (options are comma delimited):
#   detect_scans - stream4 will detect stealth portscans and generate alerts
#                  when it sees them when this option is set
#   detect_state_problems - detect TCP state problems, this tends to be very
#                           noisy because there are a lot of crappy ip stack
#                           implementations out there
#
#   disable_evasion_alerts - turn off the possibly noisy mitigation of
#                            overlapping sequences.
#
#
#   min_ttl [number]       - set a minium ttl that snort will accept to
```

```
#                       stream reassembly
#
#  ttl_limit [number]     - differential of the initial ttl on a session
#                           versus the normal that someone may be playing  games.
#
#  keepstats [machine|binary] - keep session statistics, add "machine" to
#                    get them in a flat format for machine reading, add
#                    "binary" to get them in a unified binary output
#                    format
#  noinspect - turn off stateful inspection only
#  timeout [number] - set the session timeout counter to [number] seconds,
#                  default is 30 seconds
#  memcap [number] - limit stream4 memory usage to [number] bytes
#  log_flushed_streams- if an event is detected on a stream this option will
#                    cause all packets that are stored in the stream4
#                    packet buffers to be flushed to disk.  This only
#                    works when logging in pcap mode!
#
# Stream4 uses Generator ID 111 and uses the following SIDS
# for that GID:
#  SID    Event description
# -----   -------------------
#  1      Stealth activity
#  2      Evasive RST packet
#  3      Evasive TCP packet retransmission
#  4      TCP Window violation
#  5      Data on SYN packet
#  6      Stealth scan: full XMAS
#  7      Stealth scan: SYN-ACK-PSH-URG
#  8      Stealth scan: FIN scan
#  9      Stealth scan: NULL scan
#  10     Stealth scan: NMAP XMAS scan
#  11     Stealth scan: Vecna scan
#  12     Stealth scan: NMAP fingerprint scan stateful detect
#  13     Stealth scan: SYN-FIN scan
#  14     TCP forward overlap

preprocessor stream4: detect_scans, disable_evasion_alerts

# tcp stream4_reassembly directive
# no arguments loads the default configuration
#  Only reassemble the client,
#  Only reassemble the default list of ports (See below),
#  Give alerts for "bad" streams
#
# Available options (comma delimited):
#  clientonly - reassemble traffic for the client side of a connection only
#  serveronly - reassemble traffic for the server side of a connection only
#  both - reassemble both sides of a session
#  noalerts - turn off alerts from the stream reassembly stage of stream4
#  ports [list] - use the space separated list of ports in [list], "all"
#              will turn on reassembly for all ports, "default" will turn
#              on reassembly for ports 21, 23, 25, 53, 80, 143, 110, 111
```

```
#               and 513

preprocessor stream4_reassemble

# http_decode: normalize HTTP requests
# ----------------------------------
# http_decode normalizes HTTP requests from remote
# machines by converting any %XX character
# substitutions to their ASCII equivalent. This is
# very useful for doing things like defeating hostile
# attackers trying to stealth themselves from IDSs by
# mixing these substitutions in with the request.
# Specify the port numbers you want it to analyze as arguments.
#
# Major code cleanups thanks to rfp
#
# unicode         - normalize unicode
# iis_alt_unicode  - %u encoding from iis
# double_encode    - alert on possible double encodings
# iis_flip_slash   - normalize \ as /
# full_whitespace  - treat \t as whitespace ( for apache )
#
# for that GID:
#  SID     Event description
# -----   -------------------
#   1      UNICODE attack
#   2      NULL byte attack

preprocessor http_decode: 80 unicode iis_alt_unicode double_encode iis_flip_slash full_whitespace

# rpc_decode: normalize RPC traffic
# ---------------------------------
# RPC may be sent in alternate encodings besides the usual
# 4-byte encoding that is used by default.  This preprocessor
# normalized RPC traffic in much the same way as the http_decode
# preprocessor.  This plugin takes the ports numbers that RPC
# services are running on as arguments.
# The RPC decode preprocessor uses generator ID 106
#
# arguments: space separated list
# alert_fragments - alert on any rpc fragmented TCP data
# no_alert_multiple_requests - don't alert when >1 rpc query is in a packet
# no_alert_large_fragments - don't alert when the fragmented
#                        sizes exceed the current packet size
# no_alert_incomplete - don't alert when a single segment
#                   exceeds the current packet size

preprocessor rpc_decode: 111 32771

# bo: Back Orifice detector
# -------------------------
# Detects Back Orifice traffic on the network.  Takes no arguments in 2.0.
#
```

```
# The Back Orifice detector uses Generator ID 105 and uses the
# following SIDS for that GID:
#  SID     Event description
# -----   -------------------
#   1      Back Orifice traffic detected

preprocessor bo

# telnet_decode: Telnet negotiation string normalizer
# ---------------------------------------------------
# This preprocessor "normalizes" telnet negotiation strings from
# telnet and ftp traffic.  It works in much the same way as the
# http_decode preprocessor, searching for traffic that breaks up
# the normal data stream of a protocol and replacing it with
# a normalized representation of that traffic so that the "content"
# pattern matching keyword can work without requiring modifications.
# This preprocessor requires no arguments.
# Portscan uses Generator ID 109 and does not generate any SID currently.

preprocessor telnet_decode

# Portscan: detect a variety of portscans
# ---------------------------------------
# portscan preprocessor by Patrick Mullen <p_mullen@linuxrc.net>
# This preprocessor detects UDP packets or TCP SYN packets going to
# four different ports in less than three seconds. "Stealth" TCP
# packets are always detected, regardless of these settings.
# Portscan uses Generator ID 100 and uses the following SIDS for that GID:
#  SID     Event description
# -----   -------------------
#   1      Portscan detect
#   2      Inter-scan info
#   3      Portscan End

# preprocessor portscan: $HOME_NET 4 3 portscan.log

# Use portscan-ignorehosts to ignore TCP SYN and UDP "scans" from
# specific networks or hosts to reduce false alerts. It is typical
# to see many false alerts from DNS servers so you may want to
# add your DNS servers here. You can all multiple hosts/networks
# in a whitespace-delimited list.
#
#preprocessor portscan-ignorehosts: 0.0.0.0

# arpspoof
#---------------------------------------
# Experimental ARP detection code from Jeff Nathan, detects ARP attacks,
# unicast ARP requests, and specific ARP mapping monitoring.  To make use
# of this preprocessor you must specify the IP and hardware address of hosts on # the same layer 2 segment as
you.  Specify one host IP MAC combo per line.
# Also takes a "-unicast" option to turn on unicast ARP request detection.
# Arpspoof uses Generator ID 112 and uses the following SIDS for that GID:
#  SID     Event description
```

```
# -----   -------------------
#   1      Unicast ARP request
#   2      Etherframe ARP mismatch (src)
#   3      Etherframe ARP mismatch (dst)
#   4      ARP cache overwrite attack

#preprocessor arpspoof
#preprocessor arpspoof_detect_host: 192.168.40.1 f0:0f:00:f0:0f:00

# Conversation
#------------------------------------------
# This preprocessor tracks conversations for tcp, udp and icmp traffic.  It
# is a prerequisite for running portscan2.
#
# allowed_ip_protcols 1 6 17
#     list of allowed ip protcols ( defaults to any )
#
# timeout [num]
#     conversation timeout ( defaults to 60 )
#
#
# max_conversations [num]
#     number of conversations to support at once (defaults to 65335)
#
#
# alert_odd_protocols
#     alert on protocols not listed in allowed_ip_protocols
#
# preprocessor conversation: allowed_ip_protocols all, timeout 60, max_conversations 3000
#
# Portscan2
#------------------------------------------
# Portscan 2, detect portscans in a new and exciting way.  You must enable
# spp_conversation in order to use this preprocessor.
#
# Available options:
#      scanners_max [num]
#      targets_max [num]
#      target_limit [num]
#      port_limit [num]
#      timeout [num]
#      log [logdir]
#
#preprocessor portscan2: scanners_max 256, targets_max 1024, target_limit 5, port_limit 20, timeout 60

# Too many false alerts from portscan2? Tone it down with
# portscan2-ignorehosts!
#
# A space delimited list of addresses in CIDR notation to ignore
#
# preprocessor portscan2-ignorehosts: 10.0.0.0/8 192.168.24.0/24
#
```

```
# Experimental Perf stats
# ----------------------
# No docs. Highly subject to change.
#
# preprocessor perfmonitor: console flow events time 10
#
##################################################
```

Todos los preprocesadores se activan en el archivo “snort.conf”, en el “paso 2”, dentro de este archivo existe una breve descripción de cada uno de ellos, y el formato general es:

preprocessor <nombre> : <Opciones_de_configuración>

A continuación se presenta cada uno de ellos:

a. Preprocesador frag2:

La fragmentación de paquetes es una técnica que emplea la familia TCP/IP para optimizar el tráfico y permitir adaptar el tamaño del mismo para pasar por distintos tipos de redes que pueden exigir un tamaño fijo o máximo de los mismos.

Una técnica conocida de ataques es justamente el aprovechamiento de esta operación. Se lleva a cabo armando el código del ataque en pequeños fragmentos, los cuales no son sospechosos pues el código maligno al completo se verá recién al rearmarse la totalidad de los fragmentos, y cada uno de ellos en sí mismo puede ser interpretado como válido. Un ejemplo de este ataque puede ser a través del empleo del programa “*fragroute*”, que hace exactamente esto. Si un IDS o FW, no llevan el control de esa secuencia, el ataque pasa de largo y se ejecuta en el destino final.

Para evitar este tipo de acciones es que se implementó este preprocesador, el cual se encarga de reensamblar los fragmentos correspondientes al protocolo IP para entregarle al motor de detección el paquete completo (y no cada uno de los fragmentos).

Un detalle de particular interés, es que una mala configuración de este preprocesador puede ser empleada para realizar un ataque DoS hacia el IDS, pues si se generan muchos paquetes fragmentados, podría llegar el caso que el IDS esté tan ocupado con todos esos reensambles que no esté en capacidad de procesar el resto del tráfico. Es por ello que este preprocesador posee parámetros que se configuran para este tipo de acciones:

- timeout [segundos] (Define el número de segundos que se almacenará en memoria un fragmento no completo a nivel IP. Superado este tiempo, se descartan todos los fragmentos

correspondientes a ese datagrama. Si no se emplea esta opción, por defecto el valor es 60 segundos).

- Memcap [bytes] (define el número de bytes disponibles en memoria para esta operación. Si no se emplea esta opción, por defecto el valor es 4MB = 4194304 bytes).
- Min ttl [número] (Define el mínimo valor de "time to life" empleado en ese fragmento IP a aceptar, cualquier valor menor a este se descarta. Si no se emplea esta opción, este valor no es tenido en cuenta).

b. Preprocesador stream4:

Este preprocesador permite llevar el control de estado de las secuencias TCP. Esta actividad tiene muchos aspectos a destacar, pues permite:

- Verificar la apertura y cierre de sesiones.
- Realizar el seguimiento de tráfico de cada sesión (es decir, entre cada cliente y servidor individualmente).
- Colaborar con la identificación de scan de puertos.

Stream4 tiene 2 características fundamentales:

- Control de estados TCP: Se basa en los flags de TCP (SYN, ACK y FIN), con los cuales se establecen y cierran sesiones y a su vez en los números que permiten realizar el secuenciamieno (Ns y Nr: concepto de ventana deslizante) y en los pasos establecidos para toda esta actividad.

- Reensamble de sesiones: permite analizar el paquete completo y no cada una de sus partes.

Este preprocesador posee varias opciones (por defecto tiene timeout=30 segundos y reserva 8Mbyte de memoria de captura), las opciones son:

- -z [est | all]: permite almacenar el estado de todas las conexiones TCP y alertar sólo cuando no se respeta la secuencia de conexión y/o desconexión adecuada.
- detect_scan: como su nombre lo indica permite detectar scan de puertos.
- detect_state_problems: problemas con estados TCP.
- disable_evasion_alerts: deshabilita la posibilidad de generar ruido de secuencias mal configuradas, para evadir otros eventos.

- Min_ttl [número]: perite configurar el valor mínimo de ttl para aceptar el reensamble de paquetes.
- Ttl_limit [número]: Diferencia máxima de ttl que puede ser tolerada entre paquetes de una misma sesión.
- Keepstats [machine | binary]: guarda estadísticas de cada sesión TCP en formato binario o texto.
- Noinspect: deshabilita el control de estados.
- Timeout [número]: configura el tiempo de almacenamiento de cada sesión.
- Memcap [número]: configura la cantidad de memoria disponible para stream4.
- Log_flushed_streams: si se detecta un evento dentro de un flujo, esto causa que sea guardado el paquete completo. Sólo opera en modo pcap.

c. Preprocesador stream4_reassembly:

Este preprocesador permite especificar cuáles sesiones se desea reensamblar. La configuración por defecto, solo reensambla los paquetes del cliente y la lista de puertos por defecto (21, 23, 25, 53, 80, 143, 110, 111 y 513). Las opciones son:

- clientonly: reensambla solo del lado cliente.
- Serveronly: solo del lado servidor.
- Both: ambos.
- Noalerts: deshabilita las alertas.
- Ports [lista]: permite especificar qué puertos reensamblar (se coloca la lista separada por espacios, o las palabras “all” o “default”).

d. Preprocesador http_decode:

Este preprocesador, normaliza el formato de las solicitudes http, substituyendo todos los caracteres del tipo %xx (unicode), iis (UTF-8), caracteres de escape, etc.. a su equivalente ASCII. Permite especificar la lista de puertos que se deseen, separados por espacios

e. Preprocesador rpc_decode:

El protocolo RPC permite que un programa de un host llame a otro programa en un host

diferente (se encuentra bien documentado en la RFC 1831). RPC se ejecuta a nivel de aplicación, y emplea 32 bits para cada registro, de los cuales el primer bit indica si el registro continúa o si es el último, y los restantes 31 bits describen el tamaño de datos del fragmento. Por encontrarse a nivel de aplicación, se puede fácilmente generar tráfico que sea fragmentado en los niveles inferiores, y de esta forma, disfrazar el contenido de cada registro, es decir, partir esos 32 bits en fragmentos diferentes. Con este método, un IDS, que no pueda "normalizar" cada uno de estos grupos de 32 bits, pasaría por alto el contenido de la información que contiene a continuación.

El preprocesador_rpc entonces, de manera similar al preprocesador anterior, normaliza el tráfico RPC, pero su actividad es el control y armado de sus característicos 4-bytes de codificación, para entregarlo agrupados al motor de detección siempre en formato 4-bytes. Acepta como argumentos, la lista de puertos sobre los que esté trabajando RPC separados por espacios.

f. Preprocesador Back Oriffice:

Este preprocesador examina todos los paquetes UDP mayores de 18 bytes, verificando los primeros 8 caracteres con (en realidad lo hace primero con 2, y luego posee un proceso que decide si continuar o no...), para verificar la existencia del criptografiado que emplea BO y lo confronta contra un tabla que Snort carga al iniciarse.

Detecta tráfico Back Orifice. No posee argumentos.

g. Preprocesador telnet_decode:

Este preprocesador, también se dedica a normalizar el tráfico, en este caso el de las secuencias de negociación de telnet, buscando paquetes que no respeten la misma. No requiere argumentos, pero se pueden especificar los puertos que se deseen, separados por espacios.

h. Preprocesador portscan:

Este preprocesador detecta los tipos de scan lanzados desde un host hacia varios puertos durante un cierto período de tiempo, por defecto lo hace sobre 4 diferentes puertos en menos de 3 segundos . Trabaja sobre scan UDP y TCP y sus flags u opciones correspondientes.

El problema que plantean los detectores de scan es el llamado "slow scan", es decir qué sucede si se realiza un scan, pero por ejemplo lanzando un paquete cada diez minutos. Nuevamente se debe decidir entre el límite de detectar y/o dejar el IDS fuera de servicio, pues para esta

actividad el IDS necesita almacenar en memoria RAM, la totalidad del tráfico dirigido hacia los diferentes puertos de la red que monitoriza, y por lo tanto, si se almacenan intervalos de tiempo muy grandes, es peligroso, y si son pequeños, no estaría en capacidad de detectar un "Slow scan".

Para configurar este preprocesador es necesario evaluar los siguientes parámetros:

- Red a monitorizar: Suele ser la variable $HOME_NET, pero puede especificarse lo que se desee en formato CIDR separado por espacios (Ej:10.0.0.0/8).
- Número de puertos: Cantidad de puertos en el intervalo de tiempo que se especifica a continuación.
- Intervalo de tiempo en segundos: Determina el umbral de tiempo en el que se scanean los puertos del punto anterior.
- Archivo donde se almacenarán los eventos.

Ejemplo: preprocessor portscan: $HOME_NET 4 3 portscan.log

i. Preprocesador portscan-ignorehosts:

Es un complemento del anterior y permite especificar qué host se desea que sean ignorados para el procesador de scan. No posee parámetros, simplemente se colocan las redes o host que se desea ignorar en formato CIDR, separados por espacios.

j. Preprocesador portscan2:

Este amplía al anterior (y se prevé que será el único a emplear a futuro). Se debe combinar con el preprocesador "conversation".

k. Preprocesador arpspoof

Permite comparar la correspondencia entre una respuesta ARP y una tabla cargada en Snort, que posea la lista de IP-MAC de la propia red. Este preprocesador actualmente se encuentra en estado experimental y puede bajar sensiblemente el rendimiento de la herramienta si la red es considerable. Se aclara aquí que existen otras herramientas que realizan específicamente esta actividad, independientemente de Snort, como puede ser ARPWATCH, que en muchos

casos debería analizarse su empleo en otro hardware diferente de donde se ejecuta Snort.

l. Preprocesador conversation:

También se encuentra en estado experimental y su actividad es la de guradar registros de cada comunicación entre dos hosts, permitiendo al preprocesador scan2 realizar el seguimiento y potencial alerta de una conversación.

6. REGLAS (o firmas):

Este es un aspecto fundamental de Snort, y que se tratará de aclarar lo mejor posible.

Una regla de Snort, se divide en dos secciones: Encabezado y Cuerpo.

a. **Encabezado:** Es la parte principal de una regla, especifica qué es lo que debería hacerse al encontrarse una correspondencia con ella, qué protocolo emplear y las direcciones y/o puertos. El encabezado se divide en cuatro categorías: Acción, Protocol, Fuente y Destino.

1) Acción: Esta le dice a Snort qué debe hacer cuando la regla se cumple. Posee cinco opciones:
   - Pass: Ignora el paquete.
   - Log: Almacena el paquete hacia dónde se haya especificado el formato Looging mode (explicado más adelante en el punto 7.).
   - Alert: Esta acción almacena igual que el formato log y a su vez también envía una alerta (en modo alerta (explicado más adelante en el punto 7.), acorde a lo configurado en el archivo "snort.conf".
   - Dynamic: Esta se emplea de manera combinada con la siguiente acción (activate), y en realidad permanece inactiva hasta que es disparada por un paquete que se corresponde con una regla "activate" que apunta a esta dinámica, a partir de ese momento, esta regla (Dynamic) opera como cualquier otra de tipo "Log".
   - Activate: Al cumplirse una regla de este tipo, lo primero que hace es generara una alerta y luego inmediatamente activa la regla dinámica a la que esté dirigida esta regla.

Aparte de estas cinco categorías, Snort ofrece también la posibilidad de programar cualquier tipo que se necesite. Esta actividad no será tratada en este texto, pero se puede buscar la metodología (que es realmente simple) en los mismos manuales de Snort.

2) Protocolo: En la actualidad, Snort soporta cuatro opciones de protocolo: IP, ICMP TCP y UDP. Para especificar el tipo de protocolo, simplemente se coloce este nombre luego de la acción y separado por un espacio. El ritmo de avance de Snort es realmente vertiginoso, así que es conveniente consultar su página web al hacer uso de la herramienta para corroborar el desarrollo de nuevos protocolos, los cuales se están probando cotidianamente.

   Solo se puede declarar un protocolo por regla (y no admite más)

.

3) Fuente: Esta es la tercera parte del encabezado y admite varias opciones, puede hacerse nombrando las variables estáticas, dinámicas o empleando la notación CIDR, permite también el uso de "any". Puede emplearse también la negación, a través del operador "!" delante de la dirección.

   Dentro del campo Fuente, se encuentra incluido también el puerto origen, el cual puede ser expresado por medio de un número (1 – 65535), o a través del nombre del protocolo al cual se está accediendo por medio de ese puerto. Otra alternativa que presenta es la de declarar o excluir rangos de puertos, se coloca el valor de inicio y el final, separardo por ":"

4) Destino: Idem a Fuente.

   Entre los campos Fuente y Destino, se emplea el operador de dirección. Este operador indica hacia que sentido se dirige el tráfico, y por lo tanto cómo será analizada la regla, las tres opciones son:

   - Fuente -> Destino
   - Fuente <- Destino
   - Fuente<-> Destino

Ejemplos de encabezados de reglas:

```
log tcp 10.10.0.0/16 any <-> 192.168.10.0/24 any
alert icmp any any      -> $HOME_NET any
alert tcp any any <-> any any
pass tcp $INTERNAL_NET any -> $EXTERNAL_NET any
alert tcp $EXTERNAL_NET any -> $INTERNAL_NET 137:139
```

b. **Cuerpo:** El cuerpo de una regla no es obligatorio, pero es donde se definen parámetros concretos

que pueden ajustar al trabajo de una regla al proceder deseado, describir la misma, los mensajes que debe enviar, etc.

El cuerpo de una regla comienza abriendo un paréntesis y finaliza cerrándolo. El cuerpo se divide en dos partes, separadas por ":": El cuerpo principal y las opciones.

- El cuerpo principal: Especifica el contenido.

- Las opciones: Consisten en una gran cantidad de parámetros opcionales que se pueden configurar.

El contenido de las opciones se puede escribir en ASCII (Se encierra entre "", y es necesario incluir ". | ´ " se encapsula entre " ") o binario (Se lo representa en su valor hexadecimal, precedido por "|").

Ejemplos:

alert tcp any any <-> any any (content: "esto es una opcion";)

alert tcp any any <-> any any (content: "|0105 0BFF|")

Existen muchas opciones que permite configurar Snort, las mismas serán tratadas a continuación:

- depth: esta opción permite especificar hasta cuantos bytes después del encabezado IP serán analizados por esta regla. Superados estos bytes, no continúa el análisis. Con esto se reduce tiempo de CPU, pues un paquete cualquiera, independientemente del tamaño que tenga, para este regla, solo contarán los byte que se hayan impuesto en la opción depth.
- offset: Esta opción permite declarar una posición determinada a partid de la cual empezará a buscar dentro del paquete.
- nocase: A través de esta opción, se puede aclarar que la regla no sea sensible a mayúsculas y minúsculas.
- session: Esta opción se emplea para guardar todos los datos de una sesión. Es muy útil en los casos en que se emplea texto plano. Se debe aclarar una de dos acciones que toma esta opción: "all" o "printable".
- uricontent: esta opción es muy útil para evitar que se analice todo el contenido de un paquete, en los casos en que solo se desea observar dentro del "URI content", es decir al emplear un path determinado dentro de la sección URI de una solicitud.

- stateless: Esta opción en las versiones actuales de Snort, queda incluida dentro de la opción flow (que se trata más adelante).
- regex: Esta opción (en estado experimental), habilita la posibilidad de emplear los conocidos comodines "?" y "*", para reemplazar uno o varios caracteres dentro de la búsqueda.
- flow: La característica importante de esta regla, es que no necesita definir el sentido del tráfico a nivel IP, sino que puede ser útil para seguir el flujo bajo un concepto Cliente-Servidor. El empleo más conocido es cuando un cliente genera tráfico malicioso hacia un servidor, se puede emplear esta opción para verificar ¿qué es? lo que el servidor responde. Se debe especificar cuál es el valor que se le asigna, separado por ":" y permite varias opciones: to_server, from_server, to_client, from_client, only_stream, no_stream, established, stateless.
- "Opciones IP": existe un gran conjunto de opciones para IP, casi todas relacionadas con el formato del encabezado IP: D, M y R (Bits fragmentación y reservados) – sameip (misma IP origen y destino) – ipopts (IP options) – tos (Tipe Of Service) – ttl (Timi TO Life)
- "Opciones TCP": también casi todas relacionadas a su encabezado.

7. SALIDAS Y LOGS:

Las salidas difieren del resto de los componentes de Snort, pues no existe un único punto de entrada para las opciones de salida (aunque parezca raro), es decir que existen diferentes partes de Snort que pueden generar salidas, estas son:

- El decodificador de paquetes: Puede ejecutarse para generar salidas tipo tcpdump o texto.
- Algunos preprocesadores: existen algunos de estos módulos, que tienen sus propias salidas (Ej: portscan).
- El motor de detección: emplea los plug-ins de salida a alert y syslog.

El primer concepto que se debe tener en cuenta es algo que ya se ha mencionado, pero vale la pena reiterar. Si se desea guardar información en formato log hacia el disco duro, se debe emplear la opción "-l" y aclarar luego el path hacia donde se dirigirá este archivo y su nombre (Ej: snort –de –l /var/log/snort_log).

En muchos casos es conveniente dejar la opción de almacenar estos datos en formato binario (también llamado formato "pcap"), lo cual es una excelente alternativa para visualizarlos con alguna herramienta de análisis de tráfico (como también se mencionó con anterioridad), para este caso se emplea la opción

"-b" (Ej: snort –de –l /var/log/snort_log –b). Otra gran ventaja de esta opción es que es extremadamente rápida. Si se desea leer este archivo, se emplea la opción "-r".

La configuración por defecto de Snort, cuando se lo emplea como IDS, es almacenar los eventos en formato denominado **"alert"** en un subdirectorio denominado **"log".**

El mejor modo de configurar múltiples salidas es hacerlo a través del paso 3 en el archivo "snort.conf", el cual se presenta a continuación:

```
###################################################################
# Step #3: Configure output plugins
#
# Uncomment and configure the output plugins you decide to use.
# General configuration for output plugins is of the form:
#
# output <name_of_plugin>: <configuration_options>
#
# alert_syslog: log alerts to syslog
# ----------------------------------
# Use one or more syslog facilities as arguments.  Win32 can also
# optionally specify a particular hostname/port.  Under Win32, the
# default hostname is '127.0.0.1', and the default port is 514.
#
# [Unix flavours should use this format...]
# output alert_syslog: LOG_AUTH LOG_ALERT
#
# [Win32 can use any of these formats...]
# output alert_syslog: LOG_AUTH LOG_ALERT
# output alert_syslog: host=hostname, LOG_AUTH LOG_ALERT
# output alert_syslog: host=hostname:port, LOG_AUTH LOG_ALERT

# log_tcpdump: log packets in binary tcpdump format
# -------------------------------------------------
# The only argument is the output file name.
#
# output log_tcpdump: tcpdump.log

# database: log to a variety of databases
# ---------------------------------------
# See the README.database file for more information about configuring
# and using this plugin.
#
# output database: log, mysql, user=root password=test dbname=db host=localhost
# output database: alert, postgresql, user=snort dbname=snort
# output database: log, unixodbc, user=snort dbname=snort
# output database: log, mssql, dbname=snort user=snort password=test

# unified: Snort unified binary format alerting and logging
```

```
# ----------------------------------------------------------------
# The unified output plugin provides two new formats for logging
# and generating alerts from Snort, the "unified" format.  The
# unified format is a straight binary format for logging data
# out of Snort that is designed to be fast and efficient.  Used
# with barnyard (the new alert/log processor), most of the overhead
# for logging and alerting to various slow storage mechanisms
# such as databases or the network can now be avoided.
#
# Check out the spo_unified.h file for the data formats.
#
# Two arguments are supported.
#    filename - base filename to write to (current time_t is appended)
#    limit    - maximum size of spool file in MB (default: 128)
#
# output alert_unified: filename snort.alert, limit 128
# output log_unified: filename snort.log, limit 128

# You can optionally define new rule types and associate one or
# more output plugins specifically to that type.
#
# This example will create a type that will log to just tcpdump.
# ruletype suspicious
# {
#   type log
#   output log_tcpdump: suspicious.log
# }
#
# EXAMPLE RULE FOR SUSPICIOUS RULETYPE:
# suspicious $HOME_NET any -> $HOME_NET 6667 (msg:"Internal IRC Server";)
#
# This example will create a rule type that will log to syslog
# and a mysql database.
# ruletype redalert
# {
#   type alert
#   output alert_syslog: LOG_AUTH LOG_ALERT
#   output database: log, mysql, user=snort dbname=snort host=localhost
# }
#
# EXAMPLE RULE FOR REDALERT RULETYPE
# redalert $HOME_NET any -> $EXTERNAL_NET 31337 (msg:"Someone is being LEET"; \
#   flags:A+;)

#
# Include classification & priority settings
#

include classification.config

#
# Include reference systems
#
```

```
include reference.config

##################################################################
```

Como se puede apreciar existen diferentes formatos de salida par snort, estos son:

- CSV (Comma Separated Values): permite fácilmente importar datos desde varias asplicaciones.
- Syslog: Similar a la opción desde “alert facility”, permite personalizar la facilidad de syslog.
- Database: Permite la salida a distintos formatos de bases de datos: MySGL, PostgreSQL, UnixODBC, Oracle y SQL Server.
- Null: Permite a Snort enviar a “alert facility”, pero no crea ningún archivo de log.
- Tcpdump: Salida en formato tcpdump.
- SnmpTrap: envía traps SNMP a un servidor SNMP.
- Unified: Es el futuro de Snort y es la salida más rápida. Emplea el formato binario con Fast alert. Emplea el programa Barnyard para leer posteriormente.
- XML: Salidas en formato SNML (Simple Network Markup Lenguage).

Existen dos “MODOS” de salida para Snort:

a. Modo alerta: Cuando el “alert” de una regla genera una alarma, se tienen en cuenta dos acciones:

- Salida del evento (denominada “facilidad”):

  La facilidad controla el formato de la alerta, algunos aspectos y su destino. Las opciones de facilidad son:

  - Full (por defecto): contenido completo de todos los encabezados.
  - Fast: Formato simple, con tiempo, mensaje de alerta, direcciones fuente y destino y puertos.
  - Syslog: Envía a syslog. (LOG_AUTH_PRIV y LOG_ALERT).
  - Unsock: Envía a un socket UNIX.
  - SMB: Mensaje winpopup de windows.
  - None: desactiva alertas.
  - Console: envía alertas en modo fast a la consola.

- Enviar un log acorde a la configuración de detalle deseada: Esto se trata en el punto siguiente (b. Modo logging).

b. Modo logging:

Almacena la información completa, pero sin generar una alerta. Este modo puede ser activado, empleando directamente la palabra "log", dynamic" o "alert" en las reglas de snort (Se debe tener en cuenta que si no se inicia Snort con la opción –l este modo sólo lo activan las "alert" de las reglas, como se trató en el apartado anterior) . Por defecto Snort lo hará siempre en /var/log/snort, pero puede ser cambiado con la opción "-l" a la ruta y archivo que se desee. Se crea un directorio por cada dirección IP que detecte Snort y dentro de cada uno de ellos se almacenarán las alertas correspondientes.

Como se mencionó con anterioridad, la mejor forma de trabajar con las salidas es a través del archivo **"snort.conf"** en el paso 3.

Las salidas se declaran con la forma: output <nombre>: <opción>

Se tratarán a continuación algunas de ellas:

1) Output log_null: Configura las salidas de Snort para que no creen los subdirectorios por cada dirección IP que escucha.

2) output alert_CSV: <filename> <format>: Esta es queizás la salida más flexible para operar posteriormente, pues permite varias opciones, las que entregará separadas por comas. Lo realmente útil es el campo <format>, en el cual se puede especificar, separados por coma, los siguientes parámetros: *_timestamp, sig_generator, sig_id, sig_rev, msg, proto, src, srcport, dst, dstport, ethsrc, ethdst, ethlen, tcpflags, tcpseq, tcpack, tcplen, tcpwindow, ttl, tos, id, dgmlen, iplen, icmptype, icmpcode, icmpid, icmpseq.* SIN DEJAR ESPACIOS!!!!!

Ejemplos:

output alert_CSV: /var/log/alert.csv default

output alert_CSV: /var/log/alert.csv timestamp,sid,proto,src,dst

3) output log_unified: filename snort.log : Se trata del método más rápido para almacenar eventos, permite hacerlo a "alert y log files", acorde al nivel de detalle que se desee. Los eventos se almacenan en formato binario, pero no es el mismo formato de pcap (es decir no se lo puede leer directamente con tcpdump o ethereal), este nuevo formato, se aprecia que será el futuro de Snort y fue pensado para trabajar con la herramienta BARNYARD (que puede descargarse de www.snort.org), la cual permite varios tipos de configuración. Otro programa más simple para transformar formatos es logtopcap.c (que puede descargarse de http://dragos.com/logtopcap.c), este programa una vez compilado (gcc –o nombre_final logtopcap.c) permite transformar rápidamente cualquier archivo en formato unificado a formato binario (pcap), se ejecuta de la siguiente forma:

./logtopcap snort.log.nnnnnnnn archivo_bin_destino

4) output log_tcpdump: almacenará paquetes "log" en formato tcpdump en la ruta especificada. Y crea archivos del tipo "snort.log.nnnnnn". Para leer estos archivos, se puede emplear tcpdump – r nombre_arch o Ethereal. Ejemplo:

output log_tcpdump: /usr/local/bin/log/snort.log

8. **PRACTICAS**

1. **salidas:**

./snort –c snort.conf (sale por defecto con formato "full", hacia /var/log/snort/alert y también crea un directorio por cada dirección IP que detecta).

./snort –c snort.conf - A fast (sale con formato "fast", hacia /var/log/snort/alert y también crea un directorio por cada dirección IP que detecta).

./snort –c snort.conf -b (sale por defecto con formato "full", hacia /var/log/snort/alert y crea un archivo en binario con nombre "snort.log.nnnnnnn").

./snort -l /var/tmp/ -c snort.conf (sale por defecto con formato “full”, hacia /var/tmp/alert y también crea un directorio por cada dirección IP que detecta).

./snort -l ./log -c snort.conf (sale por defecto con formato “full”, hacia el directorio local /log y también crea un directorio por cada dirección IP que detecta). (DEBE EXISTIR EL DIRECTORIO LOG, sino dará un error)

2. confección de reglas:

Ejemplos de local.rules:

alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"Anonymous FTP enabled {nessus}"; flags:A+; content:"USER null"; nocase; depth: 10; reference:CVE, CAN-1999-0452; classtype:attempted-user; sid:1001001; rev:1;)

#

alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"Linux FTP Backdoor {nessus}";flags: AP; content:"PASS null"; nocase; depth: 10; reference:CVE, CAN-1999-0452; classtype:attempted-user; sid:1001015; rev:1;)

#

alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"Writeable FTP root {nessus}"; flags:A+; content:"STOR nessus_test"; depth: 20; reference:CVE, CAN-1999-0527; classtype:attempted-user; sid:1001002; rev:1;)

#

alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:" writeable FTP root {Comando CWD / - nessus}"; flags:A+; content:"CWD /"; depth: 10; reference:CVE, CAN-1999-0527; classtype:attempted-user; sid:1001003; rev:1;)

#

alert tcp $EXTERNAL_NET any -> $HOME_NET 513 (msg:"rlogin {nessus}"; flags:A+; content:"root"; nocase; depth: 10; reference:CVE, CAN-1999-0651; classtype:attempted-user; sid:1001004; rev:1;)

#

alert tcp $EXTERNAL_NET any -> $HOME_NET 25 (msg:"EXPN and VRFY commands {nessus}"; flags:A+; content:"HELO nessus.org"; nocase; depth: 20; reference:CVE, CAN-1999-0531; classtype:successful-recon-largescale; sid:1001005; rev:1;)

#

alert tcp $EXTERNAL_NET any -> $HOME_NET 25 (msg:"SMTP Server type and version {nessus}"; flags:A+; content:"HELP"; depth: 10; classtype:network-scan; sid:1001006; rev:1;)

#

alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"WFTP login check {nessus}"; flags:A+; content:"bogusbogus"; depth: 25; reference:CVE, CAN-1999-0200; classtype:attempted-user; sid:1001007; rev:1;)

#

alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:"IIS 5.0 PROPFIND Vulnerability {nessus}"; flags:A+; content:"PROPFIN"; depth: 25; classtype:attempted-user; sid:1001008; rev:1;)

#

alert tcp $EXTERNAL_NET any -> $HOME_NET 513 (msg:"SysV /bin/login buffer overflow (rlogin) {nessus}"; flags:A+; content:"nessus"; depth: 10; reference:url, www.cert.org/advisories/CA-2001-34.html; classtype:attempted-user; sid:1001009; rev:1;)

#

alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP Service Allows Any Username {nessus}"; flags:A+; content: "user pp * pass pp"; regex; nocase; depth: 20; classtype:attempted-user; sid:1001010; rev:1;)

#

alert tcp $EXTERNAL_NET any -> $HOME_NET 515 (msg:"lpd, dvips and remote command execution {nessus}"; flags:A+; content:"|F702 0183 82C0 1C3B 0000 0000 03E8 1B20 5463 5820 6F75 7470 7574 2032 3030| "; depth: 30; reference:CVE,CAN-2001-1002; classtype:attempted-user; sid:1001011; rev:1;)

#

alert tcp $EXTERNAL_NET any -> $HOME_NET 25 (msg:"SMTP antivirus filter {nessus}"; flags:A+; content:"HELO nessus"; nocase; depth: 15; classtype:attempted-recon; sid:1001012; rev:1;)

#

alert tcp $EXTERNAL_NET 20 -> $HOME_NET 8888 (msg:"BenHur Firewall active FTP firewall leak {nessus}"; classtype:attempted-recon; sid:1001013; rev:1;)

#

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 443 (msg:"EXPERIMENTAL WEB-MISC OpenSSL Worm traffic"; content:"TERM=xterm"; nocase; classtype:web-application-attack; reference:url,www.cert.org/advisories/CA-2002-27.html; sid:1001014; rev:1;)

#

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 443 (msg:"deteccion 2 WEB-MISC OpenSSL Worm traffic"; content:"|4745 5420 2F20 4854 5450 2F31 2E30 0D0A 0D0A|"; flags: AP; classtype:web-application-attack; reference:url,www.cert.org/advisories/CA-2002-27.html; sid:1001016; rev:1;)

#

alert udp $EXTERNAL_NET any -> $HOME_NET 137 (msg:"Using NetBIOS to retrieve information from a Windows host {nessus}"; content:"|0000 0001 0000 0000 0000 2043 4b41 4141 4141 4141 4141 4141 4141 4141 4141|"; depth: 33; classtype: attempted-dos; sid: 1001017; rev:1;)

#

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"gusano referente a lsass.exe"; content:"lsass.exe"; offset: 40; depth: 50; classtype:web-application-attack; sid:1001017; rev:1;)

alert tcp any any -> $HOME_NET 24 (msg: "aaaaaaaaa"; content: "a*sh"; regex; classtype: web-application-attack; sid: 1001018; rev: 1;)

#

Alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"Alibaba 2.0 buffer Overflow {nessus}"; content: "POST XXXXXXXX"; depth: 15; classtype:web-application-attack;reference:CVE,CAN-200-0626; sid:1001019; rev:1;)

#

NO FUNCIONA Y NO SE PORQUE Grrr... Alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"+ + + ATH0 modem hangup {nessus}";content:"|00|"; itype: 8; icode: 0; reference: CVE,CAN-1999-1228; classtype:attempted-dos; sid: 1001020; rev:1;)

#

Alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"Axent Raptor DoS {nessus}"; tos: 123; id:1234; ttl: 255; ip_proto: 6; reference: CVE,CVE-1999-0905; classtype: attempted-dos; sid:1001021;rev:1;)

#

Alert tcp $EXTERNAL_NET any -> $HOME_NET 514 (msg:" possible rsh scan with null username {nessus}"; content:"|0204|"; flags: S; reference: CVE,CVE-1999-0180; classtype: attempted-recon; sid:1001022;rev:1;)

#

Alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"NT IIS Malformed HTTP request Header DoS Vulnerability {nessus}"; content:"|686f 7374 6e61 6d65 203a 2058 5858 5858|"; depth:20; reference: CVE,CVE-1999-0867; classtype: attempted-dos; sid:1001023; rev:1;)

#

Alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"pimp {nessus}"; id: 69; ip_proto: 2; ttl: 255; content: "|5858 5858 5858 5858 5858|";offset: 10; depth:15; classtype: attempted-dos; sid:1001024; rev:1;)

#

Alert tcp $EXTERNAL_NET any -> $HOME_NET 25 (msg:"Sendmail redirection check {nessus}"; content:"|5243 5054 2054 4f3a 2072 6f6f 7440 686f 7374 3140|"; classtype: attempted-recon; sid:1001025; rev:1;)

#

Alert tcp $EXTERNAL_NET any -> $HOME_NET 25 (msg:"Sendmail from piped program {nessus}"; content:"|4d41 494c 2046 524f 4d3a 207c 7465 7374 696e 67|"; classtype: attempted-recon; sid:1001026;rev: 1;)

#

Alert tcp $EXTERNAL_NET any -> $HOME_NET 25 (msg:"Sendmail mailing to files {nessus}"; content:"|5243 5054 2054 4f3a 202f 746d 702f 6e65 7373 7573 5f74 6573 74|"; classtype: attempted-recon; sid:1001027;rev:1;)

#

Alert tcp $EXTERNAL_NET any -> $HOME_NET 25 (msg:"Sendmail mailing to programs {nessus}"; content:"|5243 5054 2054 4f3a 207c 7465 7374 696e 67|"; classtype: attempted-recon; sid: 1001028; rev:1;)

PARTE B: NESSUS 2.0.0

1. Introducción:

Nessus es una herramienta de auditoría de seguridad de código abierto, puede descargarse y realizar cualquier tipo de consultas en:

www. Nessus.org

Consta de dos partes, un servidor (nessusd) que es quien lanza los ataques y un cliente que es la interfaz gráfica con el usuario.

Nessusd inspecciona el host los hosts remotos e intenta descubrir todas sus vulnerabilidades o errores de configuración

2. Forma de operar:

Nessus puede ser operado desde línea de comandos o desde la consola gráfica:

a. desde la consola:

Se lanza primero el demonio con ./nessusd y posee una serie de opciones:

-c <archivo_configuración>: por defecto es el archivo nessusd.conf (en una instalación estándar se ubicará en: /usr/local/etc/nessus), pero puede emplearse cualquier otro archivo de configuración.

-a <dirección>: Le indica al servidor que solo escuche conexiones sobre esa dirección IP. Se suele emplear para definir una dirección de administración y negar la posibilidad que alguien pueda ejecutar nessus desde la interfaz que está conectada haca el exterior.

-p <puerto>: Le indica al servidor que solo escuche conexiones a través de ese puerto. Por defecto es el 1241.

-D: Se ejecuta el servidor en segundo plano (Background).

-v: Indica la versión.

-h: help.

Si se va a operar en modo comando es aconsejable lanzar el demonio con & o –D (./snort &, o ./snort –D), para que se ejecute en modo Background y deje la consola abierta.

La opción –help proporciona todos los detalles de estas opciones, como se puede apreciar a continuación:

```
[root@ale sbin]# nessusd --help
    nessusd, version 2.0.0

    usage : nessusd [-vcphdDLC] [-a address]

        a <address>    : listen on <address>
        v              : shows version number
        h              : shows this help
        p <number>     : use port number <number>
        c <filename>   : alternate configuration file to use
                         (default : /usr/local/etc/nessus/nessusd.conf)
        D              : runs in daemon mode
        d              : dumps the nessusd compilation options
```

Una vez lanzado el demonio, se trabaja con el cliente, que puede ser también en modo gráfico y en modo línea de comandos. Para operar con línea de comandos, existen varias opciones, que también se pueden consultar con la opción –help, como se muestra a continuación:

```
        [root@ale sbin]# nessus --help
        nessus, version 2.0.0

        Common options :
         nessus [-vnh] [-c .rcfile] [-V] [-T <format>]
        Batch-mode scan:
         nessus -q [-pPS] <host> <port> <user> <pass> <targets-file> <result-file>
        List sessions  :
         nessus -s -q <host> <port> <user> <pass>
        Restore session:
         nessus -R <sessionid> -q <host> <port> <user> <pass> <result-file>
        Report conversion :
         nessus -i in.[nsr|nbe] -o out.[html|xml|nsr|nbe]

        General options :
```

v : shows version number
h : shows this help
n : No pixmaps
T : Output format: 'nbe', 'html', 'html_graph', 'text', 'xml', 'old-xml' 'tex' or 'nsr'
V : make the batch mode display status messages to the screen.
x : override SSL "paranoia" question preventing nessus from checking certificates.

The batch mode (-q) arguments are :
host : nessusd host
port : nessusd host port
user : user name
pass : password
targets : file containing the list of targets
result : name of the file where nessus will store the results
-p : obtain list of plugins installed on the server.
-P : obtain list of server and plugin preferences.
-S : issue SQL output for -p and -P (experimental).

EJEMPLOS (Nessus cliente y servidor en modo consola):

./nessusd -v -a 10.64.130.195 -D *(ejecuta el demonio en background)*

./nessus -q localhost 1241 nessus nessus host_dest_nessus_ejemplo resultados

(ejecuta el cliente en bacground, se conecta al servidor localhost por el puerto 1241, USER:nessus, PASSWORD:nessus, obtiene los rangos a monitorizar de host_dest_nessus_ejemplo y guarda los resultados en el archivo "resultados" con formato estándar).

./nessus -q localhost 1241 nessus nessus host_dest_nessus_ejemplo resultados -c /usr/local/etc/nessus/nessusd.conf -T text

(idem anterior, pero especificando el archivo de configuración y con salida en formato texto)

b. Desde la interfaz gráfica:

Al ejecutar ./nessus sin ningún parámetro (sin –q, que habilita el modo batch), se abre la consola cliente. La misma por defecto, habilita el primer señalador que es el que se trata de "Nessud host", a través del cual se ejecuta el primer paso para poder operar con esta herramienta, es decir, lo mismo que se hace en modo comando, identioficar, donde está el servidor (Nessusd Host), el cual si es en modo local ya a parecerá por defecto "localhost", el puerto, que por defecto es 1241, el loguin y la password (los cuales fueron configurados durante la instalación). Una vez ingresado estos valores, esta interfaz ejecuta el servidor (nessusd) y se habilitan todas las aplciones gráficas.

De todas las opciones, en este texto, se tratarán solo "Plugins" y "Target", que son las que hacen falta para desarrollar la metodología de trabajo con Snort y la Matriz de seguridad.

1) Plugins: desde aquí se activan las reglas que se lanzarán durante la monitorización. Estas reglas se programan en un lenguaje particular de Nessus, llamado"NASL", el cual no interesa desarrollar aquí, pero todo aquel que haya trabajado con "C" lo entenderá bastante bien.
2) Preferencias: Dentro de este apartado, se pueden configurar todas las opciones de monitorizacion que se deseen, técnicas de scan a emplear, ping, nmap, etc. Durante este texto, se desactivarán todas, pues se trabajara escencialmente con los plugins
3) Scan Options: Aquí se puede entrar en el detalle de los puertos y tipos de scan a realizar.
4) Target Selection: Permite especificar la dirección o rango de direcciones IP a auditar, como así también la forma de almacenar los informes

Al lanzar las auditorías desde la interfaz gráfica, se abre otra ventana en la cual muestra el avance de la misma y finalizada esta actividad, se abre la ventana del informe final, donde se puede ir desplegando en distintos cuadros dentro de esta ventana, la totalidad de las novedades detectadas. Estos informes pueden ser guardados en distintos formatos y en la ubicación que se desee.

3. Metodología de trabajo con Nessus:

La interfaz gráfica de Nessus, está explicada con bastante detalle en cada uno de sus apartados (en la misma interfaz), por lo tanto, la metodología de trabajo aquí será eminentemente práctica, para poder avanzar hacia el tema final del curso que es poder operar y encontrar la correspondencia entre Nessus y Snort, que es lo que se considera fundamental.

Por lo tanto se realizaran las siguientes actividades practicas:

Activación de consola.

Selección de plugins.

Selección de Scan.

determinar el lanzamiento de plugins puntuales.

Cuando "Conecta" y cuando "no" un plugins

Entender los formatos de .NASL (en los plugins)

Determinar cuál es el patrón que busca o que lanza.

Visualización de informes.

PARTE C: Metodología de trabajo NESSUS - SNORT

PRIMERA PRÁCTICA A REALIZAR:

Lanzar snort (definir bien variables, preprocesadores, salidas y reglas)
Abrir la interfaz gráfica de Nessus y conectarse.
Verificar el buen funcionamiento de ambos procesos
Seleccionar un plugin de Nessus y lanzarlo.
Analizar sus resultados.
Compararlos con lo detectado por Snort.
Conclusiones.

SEGUNDA PRÁCTICA A REALIZAR:

Se lanzarán ataques con hping2 (por parte del docente).
Verificar la respuesta de snort
Evaluar resultados.

Se activará tcpdump
se lanzarán los ataques nuevamente
se detendrán las capturas.
Se analizarán con Ethereal.
Se evaluarán los patrones de tráfico.
Se generarán las local.rules necesarias para detectarlo.
Se repetirán los ataques.
Se evaluarán resultados.

TERCERA PRÁCTICA A REALIZAR:

Se lanzarán ataques puntuales con nessus (por parte del docente, estos ataques no serán detectados por SNORT).

Verificar la respuesta de snort

Evaluar resultados.

Se explicará el plugins puntual de cada ataque.

Se analizará cad plugins (pero sin la captura con tcpdump del tráfico).

Se evaluará el formato de la regla .nasl de Nessus

Se generarán las local.rules necesarias para detectarlo.

Se repetirán los ataques.

Se evaluarán resultados.

CUARTA PRÁCTICA A REALIZAR:

Se lanzarán ataques puntuales con Nessus (por parte del docente, estos ataques no serán detectados por SNORT).

SE APLICARÁ LA INICIATIVA PARA DETERMINAR CÓMO HACER PARA DETECTARLO CON SNORT

Se generarán las local.rules necesarias para detectarlo.

Se repetirán los ataques.

Se evaluarán resultados.

PARTE D: MATRIZ DE ESTADO DE SEGURIDAD:

La teoría de este punto se desarrollo en otro documento denominado: “MATRIZ DE ESTADO DE SEGURIDAD_v03.pdf”, y a continuación se mencionan las prácticas que se realizarán con el mismo:

Se proporcionará cierto grado de información para completar la matriz.

Se llenarán todos los campos.

se evaluará el estado de seguridad de la red.

se definirán las acciones a seguir.

Se evaluará la misma un mes después.

Se tratará de avanzar en el tiempo, comparando la evolución de la seguridad, en base a las acciones tomadas.