Introducción a conceptos de IDS y técnicas avanzadas con Snort

Alejandro Gramajo
Baicom Networks

23 de junio de 2005

"Your eyes can deceive you. Don't trust them!
Stretch out with your feelings..."
Obiwan Kenobi

Índice

1. Introducción a IDS

La detección de intrusos es el proceso de monitorear computadoras o redes, para detectar entradas no autorizadas, actividad o modificación de archivos.

Un IDS puede también ser usado para monitorear trafico, por ej. así puede detectar si un sistema esta siendo un objetivo de un ataque de red como un DoS (denial of service).

1.1. Tipos de IDS

Existen dos grupos importantes en los tipos de IDS:

- Host IDS
  Orientados a examinar los datos en computadoras individuales.
- Network IDS
  Examinan el intercambio de datos entre computadoras. Esos paquetes son examinados y a veces comparados contra datos empíricos para verificar su naturaleza: mala o buena.

  Como se tiende a examinar varios puntos de la red es común encontrarlos de forma distribuida en la red.
- Hybrid IDS
  Una mezcla de los dos tipos anteriores.

1.2. Técnicas de detección

Existen varios técnicas y tipos de detección para los IDS, estos son los mas usados y destacados:

- Anomalía (anomaly)
  Sirve para descubrir patrones anómalos comparándolos con los considerados normales.

Especial interés para aplicación de algoritmos genéticos, redes neuronales y estadística (data mining en general [1])

Se puede separar también en "Protocol Modelling", que buscan anomalías en los protocolos o configuraciones poco comunes.

- Firma (signature)
  Comparación de firmas almacenadas contra una porción de un paquete de red.

  Reducción del análisis, subconjunto del total.

- Objetivo (target)
  Generalmente a nivel de sistema operativo, buscan modificaciones en archivos específicos, controles de accesos, uso de recursos. (HIDS en su mayoria)

1.3. Problemas con los IDS

Cuando usamos los IDS nos encontramos con dos problemas muy importantes:

- Falsos positivos
  Un falso positivo es un termino aplicado a un fallo de detección en un sistema de alertas. Sucede cuando se detecta la presencia de una intrusión en el sistema que realmente no existe.

  Principalmente se pueden dividir en 5 categorías (existen mas):

  - Reactionary traffic alarms
    Se detecta un comportamiento sospechoso como consecuencia de trafico generado anteriormente (generalmente no malicioso). Ej. ICMP network unreachable.
  - Equipment-related alarms
    Se detectan paquetes dentro del trafico de la red que identifica como no "usuales". Ej. balanceadores de carga.
  - Protocol violations
    Se producen por software mal programado (bugs) o que implementan de forma incorrecta o anticuada algunas partes de los protocolos de Internet.
  - Non malicious alarms
    Alarmas producidas al detectar rastros de comportamientos maliciosos pero que en este contexto determinado no lo son.

    Ej. publicamos un shellcode en nuestra pagina web :-)
  - True false positives
    Toda alarma que no se encuentre en las anteriores categorías.

- Falsos negativos
  Un falso negativo es un termino que hace referencia a un fallo en el sistema de alerta.

  Sucede cuando una intrusión existe en nuestro IDS y es "permitida" (ignorada o no detectada) por el sistema de alerta.

  Ej. 0 days :-(

1.4. Resumiendo

Un sistema de intrusos para que sea útil y confiable debe producir los mínimos falsos positivos posibles y "ningún" falso negativo.

2. Snort

2.1. Open source

Snort es rápido, flexible y un NIDS open-source. Empezó a fines de 1998 como un sniffer. Con licencia GPL version 2. [1]

Por default utiliza técnicas de detección de firmas y anomalías no estadística.

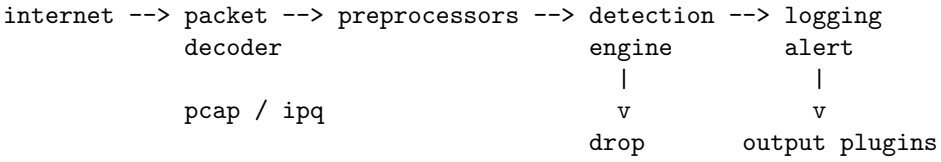
Puede correr en varios modos de ejecución:

- Sniffer (similar al tcpdump [11])
- Packet Logger
- NIDS
- IPS con FlexResp o Inline
  Requiere libnet [13]. Para Inline se necesita libipq [12].

El "engine" del Snort esta dividido en componentes:

- Decodificador del paquete (Packet Decoder)
  Toma los datos de libpcap [11] o libipq.
- Preprocesadores (Preprocessors o Input Plugins)
- Motor de detección (Detection Engine)
  Comparación contra firmas
- Logging y sistema de alerta (Logging and Alerting System)
- Plugins de salida (Output Plugins)

[1] puede cambiar a una diferente.



Estos componentes trabajan juntos para detectar ataques particulares y generar una salida en algún formato requerido.

El "decodificador de paquete", toma los paquetes de diferentes tipos de interfaces de red, y prepara el paquete para ser "preprocesado" o enviado al motor de detección.

Los "preprocesadores" son componentes o plugins que pueden ser usados con Snort para arreglar, rearmar o modificar datos, antes que el "motor de detección" haga alguna operación para encontrar si el paquete esta siendo enviado por un intruso.

Algunos preprocesadores realizan detección buscando anomalías en los headers de los paquetes y generando alertas. Son muy importantes porque preparan los datos para ser analizados contra reglas en el motor de detección.

El "motor de detección" es la parte mas importante de Snort, es responsable de detectar si alguna actividad de intrusión existe en un paquete. El motor utiliza las reglas para este propósito. Las reglas (o cadenas) son macheadas contra todos los paquetes. Si un paquete machea una regla, la acción descripta en la misma es tomada.

Dependiendo que detecte el motor dentro de un paquete, el "logging y sistema de alerta", se encarga de loguear o generar una alerta. Los logs son almacenados en archivos de texto, archivos con formato "tcpdump" u otro formato.

Los "plugins de salida" toman la salida del "sistema de alerta" y permiten almacenarlas en distintos formatos o reaccionar antes el mismo. Por ejemplo:. enviar emails, traps SNMP, syslog, insertar en una base de datos, etc. [2]

Snort nos permite extenderlo y agregarle nuevas funciones. Posee mucha facilidad para programar nuevos add-ons, ya sean preprocesadores o output plugins.

2.2. Add-ons

- Preprocesadores Existen mas de 14 pre-procesadores. Los mas destacados son:
  - port scanning (flow-portscan y sfportscan)

[2] SnortSam es un plug-in de salida y nos permite bloquear conexiones en router y firewalls.

- frag2 (ip packet defragmentation)
- stream4 (tcp stream reassembly y stateful inspection)
- http-inspect
- arp-spoof

Estos no están integrados por default el source del Snort:

- spp-fnord (Multi-architecture mutated NOP sled detector)
- Spade [19] (Statistical Packet Anomaly Detection Engine)

- Output plugins
  - Database (mysql, postgres, etc)
  - Syslog
  - XML
  - Traps SNMP
  - Mensajes SMB

2.3. Mejorando la performance

Cuando analizamos tráfico en tiempo real, necesitamos mas recursos de de "cpu" y "memoria". Obviamente depende como almacenamos las alertas la entrada y salida "io" de "disco" también puede ser un cuello de botella.

La creación de reglas óptimas es importante, para no demorar al motor de detección. En la sección 3 "Creando nuevas reglas" se dan tips para aumentar la eficacia y velocidad de las reglas.

Acá hay un detalle de dos formas de aumentar la performance de Snort.

2.3.1. MMAPed pcap

Es una reimplementación de la libpcap [9], agregándole un cambio, en vez de copiar los paquetes de la "kernel memory" a‘ "userland memory" la libpcap puede encolar los paquetes en un "shared buffer" que el Snort puede leer directamente.

Este cambio aumenta la velocidad del Snort, limitando el numero de veces que un paquete es copiado antes que el Snort pueda realizar una detección sobre el. (packet decoder)

El buffer en ethernet puede tener hasta 52Mbytes.

2.3.2. Barnyard

El método de output plugin mas rápido es el "unified", este loguea eventos en formato binario y permite guardar dos tipos de archivos, "alert" y "log". [10]

El primero tiene los datos del evento y el segundo el paquete en forma detallada.

Para recolectar esta información, se puede utilizar el Barnyard [10], es daemon que se encarga de tomar la información y enviarla a una base de datos, archivo, etc.

2.4. Software contrib

Existen varios software para administrar los IDS, mirar alertas, etc.

- ACID / BASE
  Es una consola para el Snort, que trabaja con la base de datos (interfase web, php, apache y mysql).

- Oinkmaster
  Permite hacer un upgrade de las reglas de forma automática (o semi).

- IDS Policy Manager
  Permite administrar las reglas en forma de políticas por sensor. La interfaz es muy potente pero esta hecha para Win32 y no es de código abierto.

- SnortCenter
  Administra las reglas y maneja remotamente los sensores (interfase web).

- Snortsam
  Plugin que permite bloquear direcciones IP en los firewalls mas conocidos:

  COMERCIALES:

  - Checkp*int Firewall-1
  - Cisc* PIX firewalls
  - Cisc* Routers (con ACLs o Null-routes)
  - F*rmer Netscreen, (ahora Juniper)
  - WatchGuard Fireb*x
  - 8signs (Win32)
  - M$ ISA Server (Win32)
  - CHX

OPEN SOURCE AND FREE:

- ipchains
- iptables (ebtables también)
- ipf (conocido en BSD)
- ipfw2 (FreeBSD 5.x)
- pf (OpenBSD packet filter)

▪ Y otros...

3. Creando nuevas reglas

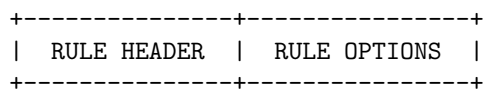
Las reglas nos sirven para machear el trafico contra una "firma", la cual dispara una alarma (o evento).

Es importante mantener nuestra base de reglas actualizada, pero para mayor eficacia es conveniente armar (o modificar) nuestras propias reglas, de esa manera minimizamos los falsos positivos.

Las reglas que abarcan muchos casos generales, suelen poseer altos índices de falsos positivos, mientras que las particulares no.

La idea es dar un detalle de como crear nuevas reglas avanzadas, en función de los servicios que deseamos monitorear.

Snort permite mucha versatilidad para crear nuevas reglas. La estructura de una regla tiene esta forma:



Y el RULE HEADER tiene una forma similar a:

ACTION	PROTOCOL	ADDRESS	PORT	DIRECTION	ADDRESS	PORT

Las RULE OPTIONS son muchísimas, seria extenso explicar todas aquí, para una referencia mayor ir al manual del Snort [1].

Ejemplos de reglas:

Generamos el mensaje "Ping" como alerta, ante la presencia de ICMP

```
alert icmp any any -> any any \
   (msg: "Ping";)
```

Si un server responde con SSH-1 tiene activado el protocolo viejo SSH1

```
alert tcp $HOME 22 -> $EXTERNAL any \
   (msg: "SSH version 1 support detected; \
   flow: to_client, established; \
   content: "SSH-1."; \
   nocase; \
   offset: 0; \
   depth: 6;)
```

Podemos activar reglas

```
activate tcp $EXTERNAL any -> $HOME 143 \
   (flags: PA; \
   content: "|E8C0FFFFFF|/bin";
   activates: 1; \
   msg: "IMAP buffer overflow";)

dynamic tcp $EXTERNAM any -> $HOME 143 \
   (activated_by 1;
   count: 50;
   msg: "50 paquetes al 143, una vez activado el IMAP bo")
```

Para una referencia detallada, de las opciones para las reglas, se debe mirar la documentación del Snort [1].

Cuando armamos una regla, para mejorar su eficacia y velocidad:

- Intentaremos machear la vulnerabilidad en si, no el código exploit, ya que estos pueden cambiar con facilidad.
- Utilizar "content" en la medida que sea posible. La primera fase del motor de detección del Snort 2.x, busca el patrón del content, cuanto mas exacto sea mas exacto el el match.
- Atrapar las singularidades del protocolo. Por ej.

  Queremos generar una alerta cuando se intenta loguear un usuario como "root" al ftp.

  ```
  alert tcp any any -> any any 21 (content:"user root";)
  ```

  Tenemos un problema ya que el protocolo ftp acepta:

  ```
  'USER root'
  'user  root'
  'user     root'
  'user<tab>root'
  ```

Una regla mejor seria:

```
alert tcp any any -> any 21 \
   (flow:to_server,established; \
   content:"root"; \
   pcre:"/user\s+root/i"; )
```

El "flow" se utiliza para verificar que el trafico esta llendo hacia el server y esta establecido.

El "content" machea la palabra "root", las reglas content son importantes ya que descarta rápidamente si no encuentra el match.

El "pcre" es para expresiones regulares, y en este caso busca el comando separado por un caracter o mas (espacio o tab), seguido del usuario, e ignora entre mayúsculas y minúsculas.

- Optimización de reglas. Por ej.

  NOTA:
  Las reglas poseen una naturaleza recursiva, si un patrón de una regla machea y si ninguna de las opciones posteriores falla, entonces se vuelve a buscar en el paquete otra vez, desde el lugar donde macheo la vez anterior. Y se repite hasta que el patrón no sea encontrado o hasta que las opciones concuerden.

  Si buscamos un paquete de size 1 byte con el ";" adentro: Pensaríamos esta regla:

  ```
  content:"|29|"; dsize:1;
  ```

  Sin embargo, por la naturaleza recursiva del Snort, si tenemos un paquete de 1000 bytes con todos ";", macheara 1000 veces!

  Para evitar esto:

  ```
  dize:1; content:"|29|";
  ```

  De esta manera solo agarra payloads con "data size" de 1 byte.

- Probar valores numéricos

  Existen dos opciones muy importantes que se incluyeron a partir de la versión 2.x: "byte_test" y "byte_jmp", las dos son útiles para armar reglas que chequeen bytes dentro de una cadena o payload.

3.1. Examinando tráfico

Cuando creamos una regla para un servicio especifico, lo mejor que podemos hacer es analizar el protocolo del mismo. Buscar como funciona, donde están las fallas o donde pueden estar.

Para eso es importante manejar alguna herramienta de "sniffing", en esto nos puede ayudar el Snort (en modo sniffer) o el TCPDUMP.

Veremos en el próximo tema, un detalle de un protocolo especifico, y como ajustar una regla para el mismo.

3.2. Event thresholding

Esta es una extensión especial a las opciones, que nos permite reducir el numero de alertas logueadas para evitar "ruido". Y puede ser usado para crear un nuevo tipo de reglas, que limitan la cantidad de veces, que se produce un evento en un determinado intervalo de tiempo.

4. Modificando y rechazando paquetes

Una utilidad que se le puede dar al Snort es transformarlo en un Intrusion Prevention System (IPS).

Un punto importante en este tema es que el Snort debe estar en un punto especial de nuestra red, ya sea como gateway/firewall o un bridge. La topología de nuestra red es completamente dependiente en este caso.

Aquí hay que tener en cuenta que si el Snort comete un "falso positivo", se bloqueara una conexión "valida".

Para lograr un IPS existen varias maneras:

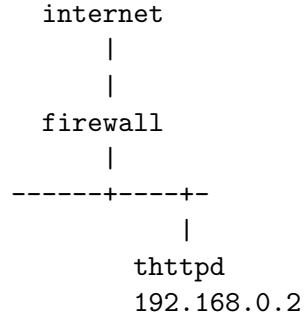
- Examinar los logs en "tiempo real" y agregar reglas en nuestro firewall (local o remoto). Como ejemplo podemos usar el SnortGuardian que utiliza nuestro firewall local (por ej. iptables/ipf/etc).

- Compilar Snort con FlexResp o INLINE. De esta manera podemos armar reglas que permitan al Snort que bloquee o rechace las conexiones, de acuerdo a los patrones que armemos. También podemos, cuando usamos "http" devolver un mensaje al cliente.

  Snort INLINE nos provee la facilidad de modificar el payload, esto nos puede servir cuando tenemos un server que no podemos patchear o el patch todavía no salio y no podemos desactivar la funcionalidad que posee el bug.

4.1. Ejemplo básico de Snort INLINE, para reescribir el payload

Supongamos que tenemos un server en la DMZ "thttpd" en el port 80, detrás de un firewall que hace NAT.



Ahora cuando vemos la respuesta del pedido "HEAD..."

```
asdf@nowhere:~$ curl -I public80
HTTP/1.1 200 OK
Server: thttpd/patched_by_baicom
Content-Type: text/html; charset=iso-8859-1
Date: Mon, 07 Mar 2005 21:04:20 GMT
Last-Modified: Fri, 29 Oct 2004 18:57:50 GMT
Accept-Ranges: bytes
Connection: close
Content-Length: 482
```

Utilizando una característica del Snort, llamada INLINE, podemos reescribir la porción de paquete que queramos, a modo de ejemplo, voy a cambiar el banner del "Server:", mas adelante haremos algo mas útil :-)

Esta seria la regla ejemplo, para reescribir:

```
alert tcp $EXTERNAL any -> $HOME 80 \
   (msg: "server signature replace"; \
   content: "Server|3A20|thttpd/patched_by_baicom"; \
   replace: "Server|3A20|thttpd/patched_by_INLINE";)
```

Acá vemos la modificación:

```
asdf@nowhere:~$ curl -I public80
HTTP/1.1 200 OK
Server: thttpd/patched_by_INLINE     <----------------- BINGO
Content-Type: text/html; charset=iso-8859-1
Date: Mon, 07 Mar 2005 21:06:40 GMT
```

```
Last-Modified: Fri, 29 Oct 2004 18:57:50 GMT
Accept-Ranges: bytes
Connection: close
Content-Length: 482
```

Bueno ahora la explicación de como funciona todo esto, pero antes a simple vista (casi) podemos ver una limitación importante que es el tamaño del paquete macheado y el tamanño del paquete modificado TIENEN QUE SER IGUALES.

Snort mas el patch de INLINE, en vez de tomar los paquetes de libpcap [11] lo toman de libipq (iptables queue).

Pasando del modo kernel al modo usuario de esta manera, en el modo usuario, en este caso dentro del Snort se le aplica un "verdict" al paquete, esto permite aceptarlo, denegarlo y modificarlo. Como vimos anteriormente.

El ejemplo que vimos recién, realmente no tiene una utilidad practica, a menos bien que queramos cambiar el banner de nuestro server web, para evitar un "banner grabbing".

Usar INLINE tiene varios problemas, si el proceso Snort, se cae, nuestro paquete no seguirá su curso, ya que necesita de un proceso en modo usuario que aplique el "verdict" correspondiente.

Otra cuestión importante es la performance, si tenemos grandes cantidades de reglas a modificar y grandes flujos de datos estamos en un problema.

Es importante recalcar que no es la única, ni la mejor forma de modificar un paquete usar Snort INLINE, pero es una opción a tener en cuenta.

4.2. Varios ejemplos mas avanzados

Para entender como armar una regla que evite falsos positivos, e intente ser lo mas óptima posible,

4.2.1. Cambiando consultas MySQL

Existen varios comandos que podríamos llamarlos "peligrosos":

```
UNION SELECT
LOAD_FILE function
LOAD DATA INFILE statement
SELECT... INTO OUTFILE statement
BENCHMARK function
User Defined Functions (UFDs)
DROP DATABASE
```

Seria interesante poder eliminarlas (hay otras formas posibles, ej. en my.cnf set-variable=local-infile=0)

Eliminamos la función "drop database":

```
alert tcp any any <> any any \
   (msg: "mysql replace"; \
   content: "drop database"; \
   replace: "select'LAMER'";)
```

Si nos conectamos al mysql desde un host remoto

```
mysql> drop database test;
+-------+
| test  |
+-------+
| LAMER |
+-------+
1 row in set (0.01 sec)
```

4.2.2. Rechazando consultas MySQL

Nos conectamos al motor, y ejecutamos un "drop table test1", poniendo el Snort en modo sniffer podemos ver el paquete que envía el cliente al server:

```
HEADER
------------------------------------------------------------------
03/10-19:07:09.572760 200.123.146.139:32975 -> 192.168.0.2:3306
TCP TTL:62 TOS:0x8 ID:12232 IpLen:20 DgmLen:73 DF
***AP*** Seq: 0x35C18013  Ack: 0xE0DAC3DA  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 156914262 165056290

PAYLOAD
------------------------------------------------------------------
11 00 00 00 03 64 72 6F 70 20 74 61 62 6C 65 20  .....drop table
74 65 73 74 31                                   test1
```

Acá podemos ver un detalle del protocolo del MySQL para "querys"::

```
11 00 00                body length= 17 bytes (little endian)
00                      packet= 0
03                      command= QUERY

64 72 6F 70 20 74 61 62 6C 65 20 74 65 73 74 31  args= drop table test1
 d  r  o  p     t  a  b  l  e     t  e  s  t  1
```

Como vemos empieza el texto del "query" en el 6to byte. Esto nos sirve para fijar el offset en 5 (empieza en 0) en nuestra regla.

Si quisiéramos rechazar la conexión:

Regla para INLINE, "reject" le dice al IPTABLES que dropee el paquete y lo loguee el Snort, y envia un TCP reset si el protocolo es TCP, o un ICMP PORT UNREACHEABLE y el protocolo es UDP

```
reject tcp $EXTERNAL any -> $HOME 3306 \
   (msg: "mysql drop reject"; \
   content: "drop table"; \
   offset: 5; )
```

Con FlexResp no necesitamos crear una regla QUEUE (no usa libipq).

Regla para FrexResp, "resp:" permite configurar modificadores de respuesta "rst_snd" envía TCP-RST al socket que esta enviando, "icmp_all" envia los paquetes ICMP_NET_UNREACH, ICMP_HOST_UNREACH e ICMP_PORT_UNREACH.

```
alert tcp $EXTERNAL any -> $HOME 3306 \
   (msg: "mysql replace2"; \
   content: "drop table"; \
   offset: 5; \
   resp: rst_snd, icmp_all; )
```

Acá vemos el efecto de la segunda regla, que utiliza FlexResp:

Nos conectamos y ejecutamos el "drop table ..."

```
bnall2:/usr/local/src# mysql -h bna140 -u test -ptest test
Your MySQL connection id is 45 to server version: 3.23.49-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> drop table test1;
ERROR 2013 (HY000): Lost connection to MySQL server during query
```

Y si miramos el paquete enviado al cliente para "resetear" la conexión...

```
03/10-19:23:13.855643 200.123.146.139:32977 -> 192.168.0.2:3306
TCP TTL:253 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
*****R** Seq: 0x71E77E7C  Ack: 0x0  Win: 0x0  TcpLen: 20
     ^
     |
     BIT RESET
```

Y podemos ver los 3 tipos de icmp devueltos...

```
03/10-20:17:02.075098 200.123.146.140 -> 200.123.146.139
ICMP TTL:110 TOS:0x0 ID:52957 IpLen:20 DgmLen:56
Type:3  Code:0  DESTINATION UNREACHABLE: NET UNREACHABLE

03/10-20:17:02.075111 200.123.146.140 -> 200.123.146.139
ICMP TTL:110 TOS:0x0 ID:52957 IpLen:20 DgmLen:56
```

```
Type:3  Code:1  DESTINATION UNREACHABLE: HOST UNREACHABLE

03/10-20:17:02.075141 200.123.146.140 -> 200.123.146.139
ICMP TTL:110 TOS:0x0 ID:52957 IpLen:20 DgmLen:56
Type:3  Code:3  DESTINATION UNREACHABLE: PORT UNREACHABLE
```

Como vemos analizando el protocolo en detalle podemos ver muchas cosas, por ejemplo si supiéramos que a partir de un tamaño de paquete, se produce un overflow, podríamos verificar el tamaño y crear alguna regla que nos avise.

Supongamos por un momento que existe un bug en los query del MySQL, y se produce cuando el tamaño del paquete es mayor a 4095 (0x0FFF)

En la primera parte del protocolo vendría algo así:

```
00 10 00    body length= 0x1000 == 4096 (little endian)
XX          packet= 0
03          command= 0x03 == QUERY
```

Hacemos una regla que si el paquete es mayor a ese tamaño dispare una alerta:

```
alert tcp $EXTERNAL any -> $HOME 3306 \
   (msg: "mysql command query packet size > 4095"; \
   byte_test:3,>,4095,0, little; \
   content: "|03|"; \
   offset: 4; )
```

4.2.3. Evitando un buffer overflow

Mirando un exploit y el reporte del: 'remote buffer overflow exploit for the Arkeia Network Backup Client' podemos armar una regla para evitarlo, veamos los detalles:

El buffer overflow ocurre cuando un paquete con tipo 77 es enviado con la sección de datos muy grande.

Para construir una regla útil, tenemos que pensar no en armar un caso, particular del código exploit, sino en entender donde esta el bug, en que parte se produce.

Primero vamos a analizar el header del protocolo enviado al cliente:

```
00 4d        2-bytes type 77 request
XX XX XX XX  4-bytes ?
nn nn        2-bytes size de la seccion datos (big endian)
...          seccion datos
```

Una regla para atrapar esta vulnerabilidad y el intento de explotarla:

```
alert tcp $EXTERNAL any -> $HOME 617 ( \
   content:"|00 4d|"; \
   offset:0; \
   depth:2; \
   byte_test:2,>,23,6; \   # bytes 7 y 8, son mayor a 23?
   isdataat:31; \          # hay datos en el byte 31?
   content:!"|00|"; \      # distinto de NULL
   offset:8; \             # a partir 9no byte
   depth:23; )             # hasta 23 bytes
```

4.2.4. Detectando un SQL injection

Para estos casos y para el XSS son muy útiles las expresiones regulares.

Acá detectamos con expresiones regulares una injección de meta-caracteres: simple-quote (’), double-dash (--) y comment (#)

```
alert tcp $EXTERNAL any -> $HTTP 80 \
   (msg:"SQL Injection - Meta chars detected"; \
   flow:to_server,established; \
   uricontent:".cgi"; \
   pcre:"/(\%27)|(\’)|(\-\-)|(%23)|(#)/i"; );
```

Detectar una ejecución de un store procedure “exec sp” o “exec xp” en M$ SQL usando como expresión regular de la alerta:

```
pcre:"/exec(\s|\+)+(s|x)p\w+/ix";
```

Para detectar comandos en particular, por ej “union”:

```
pcre:"/((\%27)|(\’))union/ix";
```

4.2.5. Detectando un XSS

Esta alerta mira un tag de apertura html (<) y su hexa equivalente, seguido de uno o mas caracteres que no son newline, y seguido de su tag de cierre (>) o su hexa equivalente:

```
pcre:"/((\%3C)|<)[^\n]+((\%3E)|>)/i";
```

4.2.6. Devolviendo un mensaje “http” al cliente

Podemos devolver un mensaje al cliente http, utiliza FlexResp:

```
alert tcp any any <> $HOME 80 \
   (content: "p0rn.html"; \ # patron que machea
   msg: "Not for you!"; \   # loguea y mensaje devuelto
   react: block, msg; )     # bloquea y envia el mensaje
```

En este ejemplo podemos usar “uricontent”. Vemos el pedido y la respuesta:

```
GET /p0rn.html HTTP/1.0

                    Snort!
                Version 2.3.0

        You are not authorized to open this site!

                    Not for you!

                    Any questions?
```

Como sugerencia, este tipo de filtros de contenido es mejor realizarlo, con un proxy server como SQUID, el cual permite redirects mas complejos.

5. Para seguir pensando

Como vemos es muy extenso el tema y la cantidad de operaciones que podemos realizar con Snort. Es importante leer y capacitarse en el uso de esta herramienta como alternativa a cualquier IDS, ya que cuenta con características altamente profesionales.

Se destaca la capacidad que tiene para extender el código y agregarle nuevas funciones, como así también quizá lo mas importante la granularidad que permite las reglas. Y no olvidarse de la posibilidad de armar un IPS usando FlexResp o Inline como vimos en los ejemplos.

5.1. Cambio de licencia para las reglas

Actualmente Snort (en realidad SourceFire) agrego un servicio con reglas “probadas” y certificadas, el cual no viene con licencia GPL, sino con una licencia limitada para su comercialización como producto o servicio.

5.2. Futuro: Detectando 0 days?

Por ahora los métodos para detectar “0 days”, son con detectores estadísticos, o analizando fallas en los protocolos.

6. Apéndice A

6.1. Compilando Snort usando Debian 3.0 (stable)

Nos traemos las libs que necesitamos

```
$ apt-get install libpcap0
$ apt-get install libpcre3-dev
$ apt-get install libpcap-dev
```

Agregamos el grupo y usuario “snort”

```
$ groupadd snort
$ useradd -g snort -s /bin/false -m -k /dev/null snort
$ mkdir /var/log/snort
$ chown snort.snort /var/log/snort/
```

Bajamos el soft

```
$ cd /usr/local/src
$ wget http://www.snort.org/dl/current/snort-2.3.1.tar.gz
$ tar -xvfz snort-2.3.1.tar.gz
$ cd snort-2.3.1/
```

Soporte MySQL

```
$ apt-get install libmysqlclient10-dev
$ ./configure --with-mysql
```

Soporte Inline

```
$ apt-get install libnet0-dev
$ ./configure \
       --enable-inline \
       --with-libipq-includes=/usr/include/libipq
```

Soporte FlexResp

```
$ apt-get install libnet0-dev
$ ./configure \
       --enable-flexresp \
       --with-libipq-includes=/usr/include/libipq
```

Y compilamos

```
$ make
$ strip src/snort
$ cp src/snort WHATEVER_YOU_WANT_DIR/
```

Referencias

[1] Snort home page: source code, manual and docs.
http://www.snort.org/

[2] Insertion, Evasion, and Denial of Service: Eluding NID.
http://www.snort.org/docs/idspaper

[3] Multi-architecture mutated NOP sled detector.
http://cansecwest.com/spp_fnord.c

[4] Securityfocus Archive IDS.
http://www.securityfocus.com/infocus/ids

[5] K. K. Mookhey and Nilesh Burghate, March, 2004.
Detection of SQL Injection and Cross-site Scripting Attacks.
http://www.securityfocus.com/infocus/1768

[6] Bleeding Snort.
http://www.bleedingsnort.com

[7] More signatures.
http://snort.demarc.com/signatures

[8] MySQL Protocol 3.2x
http://www.redferni.uklinux.net/mysql/MySQL-323.html

[9] MMAPed pcap.
http://public.lanl.gov/cpw/

[10] Barnyard.
http://www.snort.org/dl/barnyard/

[11] Libpcap, Tcpdump.
http://www.tcpdump.org/

[12] Libipq, Netfilter.
http://www.netfilter.org/

[13] The Libnet Packet Construction Library.
http://www.packetfactory.net/libnet/

[14] Gabriel Verdejo Alvarez
Sistemas de detección de intrusos (IDS)
http://tau.uab.es/ gaby/

[15] Rafeeq Rehman
Intrusion Detection with Snort: Advanced IDS Techniques Using Snort, Apache, MySQL, PHP, and ACID. ISBN 0-131-40733-3.

[16] Kerry J. Cox, Christopher Gerg
Managing Security with Snort and IDS Tools.
ISBN 0-596-00661-6.

[17] Chris Anley
Hackproofing MySQL
http://www.ngssoftware.com/papers/HackproofingMySQL.pdf

[18] William Metcalf
Snort Inline.
http://snort-inline.sourceforge.net/

[19] SPADE Statistical Packet Anomaly Detection Engine
http://www.computersecurityonline.com/spade/

[20] Defeating Honeypots: Network Issues, Part 2.
http://www.securityfocus.com/infocus/1805

[21] Data Mining in Intrusion Detection.
http://www.sans.org/resources/idfaq/data_mining.php

[22] Intrusion Detection FAQ.
Can you explain traffic analysis and anomaly detection?
http://www.sans.org/resources/idfaq/anomaly_detection.php

Licencia

Ideas para la conferencia

La idea es disertar en profundidad sobre los temas mas avanzados de este paper:

Una introducción al tema, junto con un review de posibilidades y problemas de usar Snort.

La minimización de falsos positivos, escribiendo reglas mas detalladas en cuanto al bug/exploit, error o protocolo, y optimización.

Transformar al IDS en un IPS, bloqueando contenido y manipulando paquetes.

Contacto

Alejandro Gramajo
Email Principal: agramajo at baicom.com
Email Alternativo: agramajo at gmail.com
PGP: http://people.baicom.com/~agramajo/pgp.asc