

# Introducción a conceptos de IDS y técnicas avanzadas con Snort



**Alejandro Gramajo**  
**BAICOM networks S.A.**

1º de Septiembre de 2005

**BAICOM**  
networks

# Temas

---

- **Introducción a IDS**
  - ◆ Tipos
  - ◆ Técnicas de detección
  - ◆ Problemas
- **Snort**
  - ◆ Open Source
  - ◆ Add-ons
  - ◆ Mayor performance
- **Creación de reglas**
  - ◆ Examinar trafico
  - ◆ Ejemplos
- **Modificando paquetes**
  - ◆ Snort INLINE
  - ◆ Ejemplos
- **Snort IDS en el Estado**
  - ◆ Ejemplos
- **Para seguir leyendo**

# Introducción a IDS

---

- Detección de intrusos es el proceso de monitorear computadoras o redes, para detectar entradas no autorizadas, actividad o modificación de archivos
- Tipos
  - ◆ Host IDS
  - ◆ Network IDS
  - ◆ Hybrid IDS

# Introducción a IDS

---

- Técnicas de detección
  - ◆ Anomaly (anomalía)
    - Descubre patrones anómalos comparándolo con los considerados normales
    - Especial interés para la aplicación de algoritmos genéticos y redes neuronales
  - ◆ Signature (firma)
    - Comparación de firmas almacenadas contra una porción de un paquete de red
    - Reducción de análisis, subconjunto del total
  - ◆ Target (objetivo)
    - Ej. Busca modificaciones en un archivo específico

# Introducción a IDS

---

- Problemas

- ◆ Existen 2 problemas muy importantes
  - Falsos positivos y Falsos negativos
- ◆ Los falsos positivos se pueden dividir en categorías
  - Reactionary traffic alarms
  - Equipment-related alarms
  - Protocol violations
  - Non malicious alarms
  - True false positives

- Resumiendo

- ◆ Un IDS para que sea útil y confiable debe producir los mínimos falsos positivos posibles y “ningún” falso negativo

# Snort

---

- Open Source
  - ◆ Utiliza técnicas de detección de firmas y anomalías no estadística
  - ◆ Gratuito y con licencia GPL v2
  - ◆ Modos de ejecución
    - Sniffer y Packet Logger
    - Network IDS
    - IPS con FlexResp o Inline

# Snort

---

- Open Source (cont.)
  - ◆ Componentes
    - Packet Decoder (toma los datos de libpcap o libipq)
    - Preprocesadores o Input plugins
  - ◆ Detection Engine
  - ◆ Logging y alertas
  - ◆ Output plugins
    - Enviar mails, SNMP, syslog, XML, MySQL, etc

# Snort

---

- Add-ons

- ◆ Facilidad para programarlos
- ◆ Existen más de 14 preprocesadores.
  - Port scanning (flow-portscan y sfportscan)
  - Frag2 (ip packet defragmentation)
  - Stream4 (tcp stream reassembly y stateful inspection)
  - Http-inspect
  - Arp-spoof

# Snort

---

- Add-ons (cont.)
  - ◆ No integrados por default
    - Spp-fnord (multi-architecture mutated NOP sled detector)
    - Spade (statistical packet anomaly detection engine)
  - ◆ Output plugins
    - Database (mysql, postgres, etc)
    - Syslog
    - XML
    - Traps SNMP
    - Mensajes SMB

# Snort

---

- Mayor performance
  - ◆ El análisis en tiempo real, requiere de muchos recursos de "cpu" y "memoria"
  - ◆ Optimización de reglas, necesidad de entender como funciona el engine
  - ◆ Dos formas de aumentar
    - Mmapped pcap (utilizacion de un "shared buffer")
    - Barnyard (utiliza el output plugin "unified")

# Creando nuevas reglas

---

- Formato de una regla
  - ◆ Rule header (action, protocol, address, port, direction, address, port)
  - ◆ Rule options

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

# Creando nuevas reglas

---

- Tips para armar reglas eficaces y veloces
  - ◆ Armar el patrón de regla para la vulnerabilidad, no el código exploit
  - ◆ Utilizar "content" siempre que sea posible
  - ◆ Atrapar las singularidades del protocolo
  - ◆ Las reglas poseen naturaleza recursiva
  - ◆ Probar valores numéricos (2.x)
    - Byte\_test
    - Byte\_jump

# Creando nuevas reglas

---

- Examinando tráfico
  - ◆ Cuando creamos una regla para un servicio específico, lo mejor que podemos hacer es analizar el protocolo del mismo. Buscar como funciona, donde están las fallas o donde pueden estar
  - ◆ Utilizar herramientas de sniffing

# Creando nuevas reglas

---

- Ejemplo
  - ◆ Queremos generar una alerta cuando se intenta loguear un usuario "root" al ftp

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

# Creando nuevas reglas

---

- Ejemplo (cont.)
  - ◆ Tenemos un problema ya que el protocolo acepta

```
USER ROOT
user root
user    root
user<tab>root
```

# Creando nuevas reglas

- Ejemplo (cont.)

- ◆ Flow: se utiliza para verificar que el trafico este yendo hacia el server y establecido
- ◆ Pcre: es para expresiones regulares
- ◆ Una regla mejor

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

# Modificando paquetes

---

- Snort Intrusion Prevention System (IPS)
  - ◆ Necesidad de ser gateway/firewall o bridge, la topología de la red es dependiente
  - ◆ Tener en cuenta que cuando Snort cometa un "falso positivo" se bloqueara una conexión "válida"
  - ◆ Un IPS puede loguearse:
    - Examinar los logs en tiempo real. (ej. SnortGuardian)
    - Utilizar FlexResp o Inline.
      - ◆ Se pueden armar reglas que bloqueen o rechacen las conexiones, de acuerdo a los patrones.

# Modificando paquetes

---

- Snort Inline

- ◆ También nos permite modificar el payload de un paquete, 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
- ◆ Limitación importante, el tamaño del paquete macheado y el tamaño del paquete modificado *deben ser iguales*
- ◆ Baja la performance

# Modificando paquetes

---

- Ejemplo: consultas MySQL
  - ◆ Existen algunos comandos “peligrosos”
    - UNION SELECT
    - LOAD\_FILE ...
    - LOAD DATA INFILE ...
    - SELECT ... INTO OUTFILE ...
    - BENCHMARK ...
    - UFD
    - DROP DATABASE ...
  - ◆ Algunos se pueden eliminar desde línea desde my.cnf, ej. `set-variable=local-infile=0`

# Modificando paquetes

- Ejemplo: consultas MySQL (cont.)
  - ◆ Utilizando Inline, eliminamos el *drop database*

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

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

# Modificando paquetes

- Ejemplo: mirando el protocolo MySQL
  - ◆ Levantamos Snort en modo sniffer y ejecutamos un *drop database test*

HEADER

```
-----  
C3/10-19:07:09.572760 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
```

# Modificando paquetes

- Ejemplo: mirando el protocolo MySQL (cont.)

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

64 72 6F 70 20   args= "drop "
 d  r  o  p

74 61 62 6C 65 20 args= "table "
 t  a  b  l  e

74 65 73 74 31  args= "test1"
 t  e  s  t  1
```

# Modificando paquetes

- Ejemplo: mirando el protocolo MySQL (cont.)
  - ◆ Usando Inline le decimos que dropee el paquete (Fig. 1)
  - ◆ Con FlexResp no utilizamos `libipq`, `rst_snd` envía TCP-RST e `icmp_all` envía paquetes *unreacheable* (Fig. 2)

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

Fig. 1

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

Fig. 2

# Modificando paquetes

- Ejemplo: mirando el protocolo MySQL (cont.)
  - ◆ Supongamos que existe un bug en los query MySQL, y se produce cuando el tamaño del paquete es mayor a 4095 (0x0FFF) (Fig. 1)
  - ◆ Hacemos una regla que si el paquete query es mayor dispare una alerta (Fig. 2)

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

Fig. 1

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

Fig. 2

# Snort IDS en el Estado

---

- Porque usarlo?
  - ◆ Es gratuito, open source, y es fácil extenderlo, y "customizarlo" para cualquier tipo de redes y servicios
  - ◆ Actualización de reglas, oficiales y no oficiales
- Quienes lo usan?
  - ◆ Empresas privadas actualmente utilizan Snort IDS como herramienta de detección primaria
  - ◆ Entonces porque el Estado no va a poder usarlo?

# Snort IDS en el Estado

---

- Necesidades y problemática
  - ◆ Capacitación, conectar un IDS en la red sin customizarlo no tiene sentido alguno
  - ◆ Know-How en networking, para entender y armar las reglas
  - ◆ Recursos para ver las alertas, tampoco sirve tener el mejor IDS configurado y no mirar los logs porque genera muchos falsos positivos
  - ◆ Actualización constante en materia de seguridad
  - ◆ Por lo general cuando los recursos no alcanzan, se busca tercerizar la seguridad interna

# Para seguir leyendo

---

- El paper original se lo pueden bajar de
  - ◆ <http://people.baicom.com/~agramajo/notes/ids2005.pdf>
- Snort docs
  - ◆ <http://www.snort.org/docs/>
- Security Focus Archive IDS
  - ◆ <http://www.securityfocus.com/infocus/ids>
- Listas de seguridad
  - ◆ Full-Disclosure, FOCUS-IDS, Vuln-Dev
- Bugs – exploits
  - ◆ <http://www.packetstormsecurity.org/>
  - ◆ <http://cve.mitre.org/cve/>
  - ◆ <http://www.osvdb.org/>

# Preguntas?

---

Gracias

[ang at baicom dot com](mailto:ang@baicom.com)

# Contacto

---

BAICOM networks S.A.  
Av. Leandro N. Alem 651 4º Piso  
(C1001AAB) Buenos Aires  
Argentina

Tel/Fax: (+5411) 5032.3366

[info@baicom.com](mailto:info@baicom.com)

<http://www.baicom.com>