しゃぶり尽くすsnort
&
hogwashも毒見してみる

presented by しかP

snort1.8.2(3)の紹介

- Martin Roeschが開発、メンテナンスを行う、オープンソースのIDS
- 通信パケットの内容と攻撃パターンを比較（パターンマッチング）し、攻撃を検出
- 機能が豊富で、柔軟な運用が可能
- 「その気になれば」拡張機能を作成可能

- ただし、運用は一工夫必要（IDS全般にいえること）

ついでにhogwashも紹介

- snort1.7のパターンマッチングエンジンを利用した、FW+NIDS
- 攻撃パターンにマッチした通信を遮断することで、ファイアウォールとして機能
- 100Mの通信にも対応（と謳っている）
- カーネルのIPスタックを使っていない
- IP非対応なカーネル上で動作させることで、直接攻撃を回避可能

snort：インストールの基本

- ./configure;make；suしてmake install。いじょ。
- configureの際、オプションで機能追加可能。

- Flexresp：通信強制遮断機能
- database：データベースでログ管理
- snmp：snmpでalertをチェック
- idmef：xml形式でログ出力
- smbalert：SMBでalertを送信。Windowsからチェック

・・・・などなど

Flexresp

- 攻撃とみなす通信内容が来たら、RSTパケットやICMP-unreachableを送りつけ、通信を遮断する
- パケットフィルタで止めたくないけど、あるパターンの通信は遮断したいときに有効（例：特定ユーザのFTPは許さない）
- ルールごとに定義する
- 一歩間違うと通常の（安全な）通信も遮断してしまうのが難点

Flexrespの定義

- ルールの中に「resp:hoge」と記述するだけ
- hogeにはrst_snd、rst_rcv、rst_all（以上TCP）icmp_host、icmp_net、icmp_port、icmp_all（以上UDP)を指定可能
- 例:shikapにはFTPを使わせない

```
alert tcp any any -> 192.168.0.1 21
   (msg:”shikap is coming”; content:”USER”;
   content:”shikap”;nocase;resp:rst_all;)
```

通信遮断の例：ftp

[shikap@kazumi shikap]$ ftp ponta

Connected to ponta.

220 ponta FTP server ready.

Name (ponta:shikap): shikap

421 Service not available, remote server has closed connection

Login failed.

No control connection for command: Transport endpoint is not connected

ftp> bye

通信遮断の例：ftp

kazumi.32850 > ponta.ftp: S 2534077680:2534077680(0)

ponta.ftp > kazumi.32850: S 2566713533:2566713533(0) ack 2534077681

kazumi.32850 > ponta.ftp: . ack 1

（ここで3way-handshakeが終了、コネクション確立）

ponta.ftp > kazumi.32850: P 1:81(80) ack 1

（ここで、ftpサーバのバナーが送信）

kazumi.32850 > ponta.ftp: . ack 81 （受け取り確認）

kazumi.32850 > ponta.ftp: P 1:14(13) ack 81

（ここで、USER shikapが送信される）

ponta.ftp > kazumi.32850: . ack 14 （ftpサーバの受け取り確認）

kazumi.32850 > ponta.ftp: R 2534077694:2534077694(0)

（で、すかさずRSTパケット送信、通信が中断）

Flexrespの注意点

- ともかく間違えて記述すると悲劇
- あいまいなルールはトラブルの元
- きちんとテストを行ってから実運用すること
- 止めたい通信のプロトコルを熟知しておくこと
- libnetが必要

- なによりも、snort1.8.2のFlexrespはソースが腐ってて、とんでもないポートにパケットを送りつける(1.8.3はOKらしい)
- しかも、RedHat7.1でコンパイルできない(1.8.3もだめみたい)

detabase

- データベースでログ管理可能
- 使用できるデータベースはpostgreSQL、mySQL、ODBCなど（oracleはまだβ）
- postgreSQLやmySQLを使って、ACIDを運用するとIDSのログ管理が楽。
- ACID:Analysis Console for Intrusion Database

trap_snmp

- alertをsnmpのtrapを使用してsnmpマネージャに報告
- ネットワーク管理にsnmpを使用している場合、一元管理が可能
- 開発にはIPAや日本企業が参加

trapの例

Nov 24 21:26:30 kazumi snmptrapd[2102]: kazumi [127.0.0.1]:

Trap system.sysUpTime.0 = Timeticks: (2137706) 5:56:17.06

.iso.org.dod.internet.snmpV2.snmpModules.snmpMIB.snmpMIBObjectssnmpTrap.snmpTrapOID.0 = OID: enterprises.10234.2.1.3.1

enterprises.10234.2.1.1.1.3.7 = "Snort! <*-.Version 1.8.3 (Build 87)"

enterprises.10234.2.1.1.1.5.7.5 = 0

enterprises.10234.2.1.1.1.6.7.5 = "== NA =="

enterprises.10234.2.1.2.1.2.7.5 = "1006604790. 66072"

enterprises.10234.2.1.2.1.4.7.5 = "ICMP PING *NIX"

enterprises.10234.2.1.2.1.5.7.5 = "Protocol: ICMP"

enterprises.10234.2.1.2.1.6.7.5 = 1

enterprises.10234.2.1.2.1.7.7.5 = "192.168.0.213"

enterprises.10234.2.1.2.1.8.7.5 = 1

enterprises.10234.2.1.2.1.9.7.5 = "192.168.0.0"

trap_snmpの定義と注意点

- snort.confに記述

trap_snmp: alert, internal, trap, -v2c

192.168.0.1, private

- snmpのコミュニティストリングは変更し、外部からコネクトされないようにしておく
- フィルタリングもしておきたい
- snmpマネージャが死んでいると、応答のなさに痺れを切らしてsnortが悶死

smb_alert

- NetBIOS経由で、Windowsにalertを知らせる
- Windowsのpopup_windowに表示できるっぽい (試してないので推測)
- 管理者の端末がWindowsの場合、便利
- ただし、sambaが必要 (smbclientを使用する)

XML（IDMEF)

- XML形式でログを出力
- XMLの構造はIDMEF（Intrusion Detection Message Exchange Format)を使用
- サーバにログを送信することも可能->ログをXML形式で集中管理することができる

XMLの定義

- snort.confで記述
- ファイルに出力する場合

output xml:log, file=output

- サーバに送信する場合

output xml:log, protocol=https
host=air.cert.org file=alert.snort
cert=mycert key=mykey.pem ca=ca.crt
server=srv_list.lst

- 送信プロトコルは、tcp、http、https、iap（Intrusion Alert Protocol、でもまだ未実装らしい）の4（3）種類。

XMLの例

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE snort-message-version-0.1 (View Source for full doctype...)>
- <file>
+ <event version="1.0">
- <event version="1.0">
-   <sensor encoding="hex" detail="full">
-   <interface>eth0</interface>
-   <ipaddr version="4">192.168.0.213</ipaddr>
-   <hostname>kazumi</hostname>
-   </sensor>
-   <signature id="366" revision="4" class="28" priority="3">ICMP PING *NIX</signature>
-   <timestamp>2001-11-24 21:26:29+09</timestamp>
-   - <packet>
-   - <iphdr saddr="192.168.0.213" daddr="192.168.0.0" proto="1" ver="4" hlen="5" len="84" ttl="64"
    csum="47235">
-   - <icmphdr type="8" code="0" csum="44299">
    <data>634D0D0008090A0B0C0D0E0F10111213141516171819 1A1B1C1D1E1F202122232425262
    728292A2B2C2D2E2F3031323334353637</data>
-   </icmphdr>
-   </iphdr>
-   </packet>
-   </event>
-   </file>
```

/var/log/snort/やsyslog

- /var/log/snort/内（変更可）にログを出力可能
- 実行時のオプション（-l）で指定

- syslogにalertを出力
- snort.confで出力を指定するか、実行時のオプション（-s）で指定する
- ただし、実行時オプションで指定すると、他のログ出力が停止してしまうので注意
- 例：

  Nov 24 21:26:29 kazumi snort: [1:366:4] ICMP PING *NIX [Classification: Misc activity] [Priority: 3]: {ICMP} 192.168.0.213 -> 192.168.0.0

その他のログ出力

- tcpdump形式でログ出力
- unified (snort独自形式) でログ出力
- CSV形式でログ出力
- UNIXのソケットに対してログ出力

- さまざまなログ形式をうまく活用すると管理が楽になることは間違いなし

snortの仕掛けどころ

- 通信が傍受できるところしか検知できない
- ネットワーク＝コリジョンドメインなら、どこにおいても検知可能
- スイッチなどでコリジョンドメインとネットワークが一致しない場合、通信が必ず通る場所に設置する（FW直前や直後など）
- 必要なら、すべてのホストに仕掛け、ログを集中管理（分散検知を行う）

ルールを書く場合の注意点

- ルールの文法自体は非常に簡単
- ただし、攻撃パターンがわかっていないとルールは書けない (これが一番難しい。攻撃されないとパターンはわからないから)
- tcpdumpやetherealなどのパケットキャプチャはルール作成の必須アイテム (攻撃ツールも、かな)
- あいまいなルールは誤検知の元
- 攻撃者の身になってルールを考える
- プロトコルも熟知しておきたい
- 一つの攻撃手段が様々な (そして大量の) パケットパターンをもつ場合、プリプロセッサを作ったほうが楽(例：IISのUNICODE　TRAVERSAL攻撃)

hogwashでFW

- hogwash＝snort1.7+Flexresp＋α
- αはルールの機能追加
- ユーザ数は『間違いなく』少ない。
- バージョンナンバーも小さく(0.01d)、ろくなREADMEもない
- 挙句の果てにconfigureもなかったりする
- でもなぜかsrc.rpmは存在する
- しかし、ネタとしては面白い‥‥不真面目モード炸裂

普通のFWと比較して

- パケットフィルタのように、まったく閉じてしまう必要はないので、故あって開かざるを得ない場合に効果がある
- ルールしだいでパケットフィルタとしても使える（アドレス、ポートだけでルールを作る）
- ルールセットで防御するので、ルールにない攻撃は止められない
- 誤検知はネットワーク障害の元

ルールの記述

- ほとんどsnort1.7と同じ
- 挙動を決める部分（ルールの先頭）だけはsnortと違っている
- 挙動はdrop、pass、alert、log（とsdrop。0.02から使えるらしい）を選べる
- 「論外な通信」「怪しいからalertは出すけど通信は許可」「安全な通信」のどれに該当するかを考えてルールを作成する

幸せな使い方

- 普通パケットレベルでフィルタするでしょ、というものは問答無用、パターンマッチなしでdrop（例：ソースがプライベートアドレスなパケット）
- パケットフィルタはできないけど、そういうのはなしにしてくれぇ、という場合はルールを書いてdrop（例：普通くるはずのない/bin/shの文字列）
- 微妙なんだけどdropするのは・・・という場合はalert（例：IISへのUNICODE TRAVERSAL）
- IPサポートをはずしたカーネル上で動かせば、FW直接攻撃を回避できる
- この設定なら、パケットフィルタより微妙にましなFWに仕上がる（気がする）

不幸な使い方

- デフォルトで「もう安心」と思ってしまう（デフォルトのルールは誤検知こそしないかもしれないけど間違いなく足りないと思う）
- がんばってルールを作ったはいいが、誤検知しまくってまともに通信できない
- ルールの更新をサボって新しい攻撃についていけない
- で、いつの間にかサイトは人の所有物になっている

と、持ち上げておいて

- RSTで止めるので、やっている事ばればれ。
- 結局カーネルのip_forwardを1にしないと動作せず
- 何よりも、パケットキャプチャの結果や動作チェックで驚いた（古いpcap使ったからかもしれないけど）

大弱点発覚

- eth0（内側）の通信内容が、eth1（外側）に漏れまくり
- eth0で通信しているのまで遮断しやがる
- それって、FWと言わない気がする

最後に

- snortやhogwashに頼ってはならない
- あくまでホストを堅牢にするのが先決
- snortは（Flexrespを除き）防御はしてくれない
- hogwashはまだ論外（バージョンから見ても明らか）
- 最後はクラッカーとの知恵比べ

余談

- バージョンが小さかろうが、configureがなかろうが、hogwashは「ネタとして」面白いと思う今日この頃
- 腐った仕様を何とかしたい今日この頃
- パターンマッチ以外に、侵入検知可能な仕組みが見つかればさらによし
- でも・・・・仕様があまりに腐り杉。pcapのバージョンのせいであってほしい
- バージョン0.02に期待してます（すでに0.02-pre6がでているがまだ未テスト）