Snort 初探

Aphyr Lee
aphyr@www.elites.org

2004.11.20
Outline

- How to IDSs detect intrusions
- Snort’s Inner Workings
- Playing by the Rules
- Conclusion
How to IDSs detect intrusions (1/6)

- Any way they can
  - Specialized trick for BackOrifice
    - Magic string: `!*QWTY?`
    - Random generator - `((holdrand = holdrand * 214013L + 2531011L) >> 16) & 0x7fff`
    - Bruce force decryption
  - Snort - `spp_bo.c/spp_bo.h` preprocessor
- Pattern-match
  - Searching network traffic for distinctive patterns
    - Alert tcp any any -> any any (msg: “RPC EXPLOIT startdx”; content: “/bin||c74604|sh”; sid: 600;)
    - Alert tcp any any -> any any (msg: “RPC EXPLOIT startdx”, content: “/bin||c74604|sh”; sid: 1281;)
  - Snort – `sp_pattern_match` detection-plugins

Ref: How ISS RealSecure Network Sensor 7.0 Detects Intrusions
How to IDSs detect intrusions (2/6)

- **Reassembly**
  - Data could span more than one packets
  - Snort -
    - IP deragment: spp_frag2 preprocessor
    - TCP reassembly: spp_stream4 preprocessor

- **TCP connection state**
  - Data is come from client or server

- Snort
  - spp_stream4 preprocessor
  - :sp_clientserver detection-plugins
How to IDSs detect intrusions (3/6)

* Protocol-decodes (Protocol-analysis)
  * Break down a packet into individual fields
    * Alert icmp any any -> any any (msg: “ICMP PING NMAP”; dsize:0; itype:8; sid:469;)
  * Snort
    * IP, TCP, UDP, ICMP decodes
    * Detection plugins: sp_icmp_code_check, sp_icmp_id_check, sp_icmp_seq_check, sp_icmp_type_check ...

* Application-layer Preprocessors/normalizers
  * Create some sort of “common” form
  * Evade rule-1
    * alert tcp any any -> any 21 (msg: “FTP CWD ~root”; content: “CWD ~root”; sid:336; flow: to_server;)
    * CWD ~root or CWD ~root
  * Evade rule-2
    * Alert tcp any any -> any 80 (msg: “WEB-CGI phf access”; content: “/phf”; sid: 886)
    * http:// example.com/cgi-bin/%2F%70%68%66?Qalias=%0Acat+/etc/password
  * Snort
    * Processors: spp_httpinspect, spp_rpc_decode, spp_telent_negotiation …
How to IDSs detect intrusions (4/6)

- State-based application-layer protocol decode
  - Understand the state of application-layer protocol
    - Command mode and data mode switching
    - Alert tcp any any -> any 25 (msg: “SMTP vrfy decode”; content “VRFY decode”; sid: 672)
  - Snort
    - spp_httpinspect, spp_rpc_decode, spp_telent_negotiation preprocessors
How to IDSs detect intrusions (5/6)

- Protocol-field pattern-match
  - Decoding all the fields in the packets restrict searches only to the valid contexts
    - alert tcp any any -> any 80 (msg: “WEB-CGI phf access”; uricontent: “/phf”; sid: 886;)
  - Snort
    - spp_httpinspect, spp_rpc_decode
- Protocol-validation, compliance-testing, RFC checking
  - Detection 0-day exploits
  - DNS query, HTTP method, SMTP …
How to IDSs detect intrusions (6/6)

- Other detection method
  - Protocol anomaly detection
    - Too long request, overflow attempt
  - Baseline anomaly detection
    - SYN flooding, UDP flooding
  - Heuristics and behavior analysis
    - Login failed times $\geq 3$ in short time
Snort’s Inner Workings (1/4)

```c
ProcessPacket()
{
    grinder();
    Preprocess();
}
```

```c
Preprocess()
{
    p->preprocessors = PF_ALL;
    Looping preprocessors();
    Detect();
}

Detect()
{
    fpEvalPacket();
}
```

```c
fpEvalPacket() {
    switch() {
        ...
    }
    fpEvalTCP()
    {
        retval = prmFindRuleGroupTcp(...); // Select PORT_GROUP
        switch(retval) {
            ...
        }
    }
    return FinalSelectEvent(...);
}
```
fpdetect.c

fpEvalPacket() { switch() {...} }

fpEvalTCP()
{
    retval = prmFindRuleGroupTcp(...); // Select PORT_GROUP
    switch(retval) { ...; fpEvalHeaderSW(); ...; }
    return FinalSelectEvent(...);
}

fpEvalHeaderSW()
{
    // multip-pattern detection engine
    mpseSearch(ontx_match,);
    // No-content rules matching
    for(...)
    {
        fpEvalOTN();
        fpEvalRTN();
    }
}
alert tcp any any → any 21 (msg: “FTP CWD ~root”; content: “CWD ~root”; flow: to_server; sid: 336;)

ListHead (Alert, Pass, Log, Activation, Dynamic)

RuleTreeNode

RuleFpList *rule_func;
Source IP;
Destination IP;
Source Port;
Destination Port;
RuleTreeNode *right;
OptTreeNode *down;

OptTreeNode

OptFpList *opt_func;
void *ds_list[64];
OptTreeNode *next;

OptFpList

void *context;
int (*OptTestFunc)(...);
OptFpList *next;

RuleTreeNode

RuleFpList *rule_func;
Source IP;
Destination IP;
Source Port;
Destination Port;
RuleTreeNode *right;
OptTreeNode *down;

OptTreeNode

OptFpList *opt_func;
void *ds_list[64];
OptTreeNode *next;

OptFpList

void *context;
int (*OptTestFunc)(...);
OptFpList *next;

RuleTreeNode

RuleFpList *rule_func;
Source IP;
Destination IP;
Source Port;
Destination Port;
RuleTreeNode *right;
OptTreeNode *down;

OptTreeNode

OptFpList *opt_func;
void *ds_list[64];
OptTreeNode *next;

OptFpList

void *context;
int (*OptTestFunc)(...);
OptFpList *next;

RuleTreeNode

RuleFpList *rule_func;
Source IP;
Destination IP;
Source Port;
Destination Port;
RuleTreeNode *right;
OptTreeNode *down;

OptTreeNode

OptFpList *opt_func;
void *ds_list[64];
OptTreeNode *next;

OptFpList

void *context;
int (*OptTestFunc)(...);
OptFpList *next;
PORT_RULE_MAP
PORT_GROUP *prmSrcPort[MAX_PORTS];
PORT_GROUP *prmDstPort[MAX_PORTS];
PORT *prmGeneric;

RULE_NODE
RULE_NODE *next;
RULE_PTR rnRuleData;
int iRuleNodeID;

PORT_GROUP
RULE_NODE *pgHead;
RULE_NODE *pgHeadNC;
RULE_NODE *pgUriHead;
void *pgPatData;
Void *pgPatDataUri;

Multi-Pattern Matching Structure

OptFpList
void *context;
void *ds_list[64];
OptFpList *next;
OptFpList *opt_func;

OptTreeNode
void *context;
int (*OptTestFunc)(...);
OptTreeNode *next;
OptFpList *next;

OptFpList
void *context;
int (*OptTestFunc)(...);
OptFpList *next;
# Playing by the Rules (1/10)

**Snort Rule Format**

<table>
<thead>
<tr>
<th>Rule Header</th>
<th>Rule Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>alert tcp $EXTERNAL_NET any -&gt; $HOME_NET 6666:7000</td>
<td>(msg:&quot;EXPLOIT CHAT IRC topic overflow&quot;;</td>
</tr>
<tr>
<td></td>
<td>flow:to_client,established;</td>
</tr>
<tr>
<td></td>
<td>content:&quot;</td>
</tr>
<tr>
<td></td>
<td>react: block, msg; tag: session, 10, seconds;</td>
</tr>
<tr>
<td></td>
<td>classtype:attempted-user; sid:307; rev:9;</td>
</tr>
</tbody>
</table>

## Meta-Data

- Non-Payload Detection Options
- Payload Detection Options
- Post-Detection Rule Options
- Meta-Data
Playing by the Rules (2/10)

* Rule Header
  * Rule Action
    * Alert – Alert and log packets
    * Log – Log packets
    * Pass – Ignore packets (note: Snort –o : Pass|Alert|Log)
    * Activate – Alert and turn on another dynamic rule
    * Dynamic – remain idle until activated by an activate rule
  * Protocols, IP Addresses, Port Numbers, Direction operator
Playing by the Rules (3/10)

- **Meta-Data Rule Options**
  - Provide related information to describe the rule
  - msg, reference, rev, classtype, priority

- **Payload Detection Rule Options**
  - Look for the data inside the packet payload and can be _inter-related_
  - Options
    - content, uricontent, byte_test, pcre
    - depth, offset, distance, within, byte_jump
    - nocase, rawbytes, isdataat
Playing by the Rules (4/10)

- **Payload Detection Rule Options**
  - **content**
    - Search specified content in the payload
    - content: “abcde” or content: “[0d0a 0921]”
  - **offset, within**
    - Where to start searching
    - Within is relative to the previous matched pattern
  - **depth, distance**
    - How far into a packet snort should be searched
    - Distance is relative to the previous matched pattern

![Diagram showing IP header, TCP header, and payload with offset and depth markers]
alert tcp $HOME_NET !21:23 -> $EXTERNAL_NET any \
(msg:"ATTACK-RESPONSES Microsoft cmd.exe banner";\ 
flow:from_server,established; \ 
content:"Microsoft Windows"; depth: 128;\ 
content:"|28|C|29| Copyright 1985-"; within:60; \ 
content:"Microsoft Corp."; distance:5; within:15;\ 
reference:nessus,11633; classtype:successful-admin; sid:2123; rev:2;)

Playing by the Rules (5/10)
Playing by the Rules (6/10)

- Payload Detection Rule Options
  - byte_test
    - Testing binary values or converting representative bytes testing
    - byte_test: <bytes_to_convert>, <operator>, <value>, <offset>, [, [relative], [big], [little], [string], [hex], [dec], [oct]]
  - byte_jump
    - Grap bytes, convert them to values and jump up that many bytes
    - byte_test: <bytes_to_convert>, <operator>, <offset>, [, [relative], [big], [little], [string], [hex], [dec], [oct], [align]]
alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"NETBIOS SMB NTLMSSP_AUTH unicode overflow attempt"; flow:to_server,established; content:"|FF|SMB|73|"; byte_test:1,>,127,6,relative; byte_jump:1,32,relative,word; byte_test:2,>,512,0,relative; )
Payload Detection Rule Options

- **rawbytes**
  - Look at the raw packet data, ignoring any decoding by preprocessing

- **uricontent**
  - Searches the NORMALIZED request URI field

- **pcre (Perl Compatible Regular expression)**
  - Using regular expression to search payload or URI field
  - pcre: ![]/regex/ismxAEGRUB
Playing by the Rules (9/10)

- Non-Payload Detection Rule Options
  - Look for non-payload data
  - IP fields: fragoffset, ttl, id, ipopts, fragbits, ip_proto
  - TCP fields: flags, seq, ack, window
  - ICMP fields: icode, icmp_id, icmp_seq
  - Flow: to_client, to_server, established, stateless, no_stream, only_stream
  - Flowbits: set, isset
Playing by the Rules (10/10)

- Post-detection
  - Triggers that happen after a rule fired
  - Logto – output to file
  - Session – extract user data from TCP sessions
  - Sesp – close sessions (reset)
  - React – close connection and send visible messages (to browser)
  - Tag – allow to log more than one packets after a rule is fired
Discussion