Intrusion Detection and Malware Analysis

Signature-based IDS

Pavel Laskov
Wilhelm Schickard Institute for Computer Science
Misuse detection systems

- **Expert systems** (NIDES, Emerald): rule-based decisions, rather slow, binary decisions only.
- **Signature matching** (Snort, Bro, Cisco Secure IDS, ISS RealSecure): pattern matching, policy scripting.
- **State transitions** (STAT suite): description of attacks by assertions over a state transition diagram.
- **Data mining**: automatic extraction of attack description rules from labeled examples.
- **Machine learning**: automatic extraction of “black-box” classifiers from labeled examples.
Snort highlights

- Initial open source release (December 1998):
  - plain sniffer, no rules
- 1.0 release (June 1999)
  - basic rules language, stateless packet processing
- 1.5 release (December 1999)
  - packet pipeline architecture used up-to-date
- 1.8 release developed by Sourcefire (mid-2001):
  - IP fragmentation and TCP reassembly
- Current release (2.9, 3.0beta):
  - highly stateful, 3000+ rules, protocol anomaly detection
Snort design criteria

- A **lightweight** intrusion detection tool
  - cross-platform portability
  - small footprint
  - easy installation and configuration
- A **simple language** for rules
- **High efficiency** and low memory and CPU consumption
  - packet-level detection (with no support for TCP stream reassembly in early versions)
  - packet filtering using BPF and rule hierarchy
- An **open source** alternative to expensive commercial IDS
2. Then it sends them through a chute to determine if they are coins and how they should roll (the preprocessor).

3. Next, it sorts the coins according to the coin type. This is for storage of quarters, nickels, dimes, and pennies (on the IDS this is the detection engine).

4. Finally, it is the administrator's task to decide what to do with the coins—usually you'll roll them and store them (logging and database storage).

 Packet sniffer interacts directly with a network card using libpcap.

 Preprocessing, detection and alert components are implemented as plugins.

 Various front-ends are available for logging (DB, Prelude meta-IDS, GUI).
- Operates in promiscuous mode: passes all traffic to OS.
- Performs basic packet filtering using BPF.
- Decodes packet headers using pointer casts.

```c
typedef struct _EtherHdr {
    u_int8_t ether_dst[6];
    u_int8_t ether_src[6];
    u_int16_t ether_type;
} EtherHdr;

/* lay the ethernet structure over the packet data */
p->eh = (EtherHdr *) pkt;
```
Plugin architecture enables dynamic plugin configuration.

Preprocessor functions:
- Stream reassembly (stream5)
- Packet defragmentation (frag3)
- Protocol decoding/normalization (HTTP, RPC, telnet, etc.)
- Alternative (non-rule) detection modes (portscan, arpspoof)

Plugin API enables development of custom preprocessor plugins.
- Rules are parsed into an internal data structure.
- Rule matching is prioritized according to matching complexity:
  - IP header rules
  - TCP header rules
  - Application protocol header rules
  - Content rules
- Multiple matches are possible: the highest priority alert is reported.
Various output modules are provided for handling alerts, e.g.:

- Writing to a log file
- Sending message to a syslog facility
- Logging to a database like MySQL or Oracle
- Generating an XML output
- Modifying configuration on routers and firewalls
- Sending SMB messages to Windows machines

GUI tools (ACID, SnortSnarf, Barnyard, SGUIL) are available for managing alerts.
- **General format**: header (options)
- **Header**: 
  - fixed format
  - present in every rule
- **Options**: 
  - variable format
  - not always necessary
- **Example**: 
  ```
  alert tcp $BAD any -> $GOOD any (flags: SF; msg "SYN-FIN scan");
  ```
**Snort rules: header**

- **General format:** action proto srcaddr srcport dir dst addr dstport
- **Example:** alert tcp $BAD any -> $GOOD any
- **Actions**
  - alert, pass, log
- **Protocols**
  - tcp, udp, icmp, ip
- **Directions**
  - -> (unidirectional), <> (bidirectional)
- **Variables, wildcards and expressions can be used, e.g.**
  - !$HOME, any, etc.
Snort rules: options

- **General format:** (keyword: value; )
- **Example:** (flags: SF; msg “SYN-FIN scan”; )
- **Basic options:**
  - content: pattern matcher
  - pcre: Perl-compatible regular expression
  - msg: alert message
  - flow: test for TCP connection state, traffic direction
- **Content options can be combined using the conjunction.**
Snort rules: advanced options

- Informational options
  - sid (Snort ID), priority, ref (reference, usually to CVE’s)

- Advanced payload options
  - byte_jump, byte_test, distance, within, depth, offset, nocase, rawbytes, uricontent

- Protocol analysis options
  - Keywords for IP, TCP, ICMP protocols

- Stateful rules
  - flowbits, threshold, flow

- Regular expressions
  - use with care: expensive!
alert tcp any any -> any any \ 
  (flow: established, to_server; \ 
   content: "foo"; msg: "detected foo");

alert tcp $EXTERNAL_NET any -> $SMTP_SERVERS 25 \ 
  (msg:"SMTP exchange mime DOS"; flow:to_server,established; \ 
   content:"charset = |22 22|"; nocase; reference:bugtraq,1869; \ 
   reference:cve,2000-1006; reference:nessus,10558; \ 
   reference:url,www.microsoft.com/technet/security/bulletin/MS00-082.m \ 
   classtype:attempted-dos; sid:658; rev:11;)

alert tcp $EXTERNAL_NET 80 -> $HOME_NET any \ 
  (msg:"EXPLOIT Netscape 4.7 client overflow"; flow:to_client,establish \ 
   content:"3|C9 B1 10|?|E9 06|Q<|FA|G3|C0|P|F7 D0|P"; \ 
   reference:arachnids,215; reference:bugtraq,822; \ 
   reference:cve,1999-1189; reference:cve,2000-1187; \ 
   classtype:attempted-user; sid:283; rev:10;)}
Snort summary

- A de-facto standard IDS in the practical security community
  - More than 3,000,000 downloads
  - About 200,000 users
- A light-weight, easily configurable IDS
- Good performance and reliability
- Moderate expressivity of rule language
- Numerous appliances available
  - Database, logging and alert interfaces
  - GUI tools
  - Intrusion prevention and firewall interfaces
  - Shared object rules: more complex functionality
Introduction of Bro at the USENIX Security Symposium (January 1998):
- TCP and fragment reassembly, scripting language, initial set of application-level protocol analyzers

First public release available for download:
- extended set of protocol analyzers including HTTP, SMTP

Protocol parser binpac (October 2005):
- generic protocol analysis tool, support for further protocols (SSL, RPC, NetBIOS)

Dynamic protocol detection (October 2006):
- services running at unusual ports can be detected

Broccoli: the Bro client communication library:
- event management for distributed intrusion detection
Bro design criteria

- Handling of large volumes of data
  - deployment on a high-speed link of up to 4Gb/s
- Separation of detection and policy mechanisms
  - necessity to adjust policy-based response to traffic volumes
  - sound software design
- Extensibility
  - adjustment to new services and kinds of attacks
- Robustness
  - no packet drops especially in the presence of attacks
Bro architecture

- **Sniffer**
  - libpcap with standard BPF filters
- **Event engine**
  - TCP stream and fragment reassembly
  - Generates meta-events for TCP and UDP protocols (e.g. connection_attempt, udp_request, udp_reply, etc.)
  - Application-specific events
- **Policy script interpreter**
  - Event handlers are written in a specialized Bro language: much more powerful than Snort signatures.
  - Policy scripts are compiled: efficiency gain!
Bro language

- **Strongly typed**
  - detection of type inconsistencies at compile time
  - guaranteed validity of variable values at run-time

- **Domain-specific features:**
  - host names
  - IP addresses
  - port numbers
Bro language: atomic data types

- Traditional data types:
  - bool, int, count (unsigned int), double, string (with length checking)

- Timing data types:
  - time
  - interval

- Networking data types:
  - port (e.g. http, 80/tcp)
  - addr
  - hostname (a list of possible addresses corresponding to a hostname)
Bro language: aggregate data types

- **Record**: a collection of elements of arbitrary types
  
  ```
  type conn_id: record {
    orig_h: addr;
    orig_p: port;
    resp_h: addr;
    resp_p: port;
  };
  
  Record fields are accessed using a $ operator:
  conn_id$orig_h
  ```

- **Table**: a set of elements indexed by an atomic type or a record containing only atomic types:
  
  ```
  table[conn_id] of ftp_session_info
  ```

- A **set** is a table whose indexation does not yield a value (but allows membership queries).
Bro language: operators

- **C-like operators:**
  
  +, -, *, /, %, !, &&, ||, ?:, <=, [], ( ), ++, -

- **Membership queries, e.g.:**
  
  [src_addr, dst_addr, serv] in RPC_okay

- **Regular expression queries, e.g.:**
  
  filename in /rootkit-1\.[5-8]/

- **Pre-defined functions:**
  
  - `fmt`: sprintf-like formatting
  - `mask_addr`: return top \( n \) bits of `addr`
  - `open, close`: open and close files
  - `getenv`: access environment variables
  - `system`: execute a UNIX shell command
const worm_types: table[string] of pattern = {
    ["Code Red 1"] = /.id[aq]\\?.*NNNNNNNNNNNNN/,
    ["Code Red 2"] = /.id[aq]\\?.*XXXXXXXXXXXXX/,
    ["Nimda"] = /\scripts/root\exe\\?\c\+tftp/ | /\MSADC/root.exe\\?\c\+dir/ | /cool\dll.*httpodbc\dll/, # 29Oct01 Nimda variant
} &redef;

event http_request(c: connection, method: string,
    original_URI: string, unescaped_URI: string,
    version: string) {
    for ( wt in worm_types )
        if ( worm_types[wt] in unescaped_URI )
            event worm_instance(c, wt);
}
Bro signature language

- Similar syntax with Snort rules
- Additional features:
  - Analyzer-specific content conditions:
    http-request-header /regular expression/
  - Dependency conditions:
    requires-signature [id]
  - Context conditions:
    eval policy_function
- Snort rules can be converted to Bro rules using snort2bro
Bro conclusions

- The leading research community IDS
  - Over 10 years development (the longest existing IDS project)
  - Extensive operation experience in LBNL / Berkeley
- High performance
  - Distributed operation mode available via Broccoli
- Elegant multi-tier design
- Powerful signature and policy scripting language
- Extensive support for application layer protocols
- Further info and download: www.bro-ids.org
T. Kohlenberg, editor.  
*Snort IDS and IPS toolkot.*  

binpac: a yacc for writing application protocol parsers.  

V. Paxson.  
Bro: A system for detecting network intruders in real-time.  

M. Roesch.  
Snort: Lightweight intrusion detection for networks.  