Automated Translation
Between Attack Languages
(Translating Snort rules to STATL scenarios)

Steven T. Eckmann

Reliable Software Group
University of California
Santa Barbara, CA 93106
http://www.cs.ucsb.edu/~rsg/STAT/

10 October 2001
Outline

- Problem
- Proposed solution
- Translating Snort to STATL
- Other translations
- Lessons learned
- Conclusions and future work
The Problem

- Developing IDS signatures is labor-intensive
- There are many signature-based IDSs
- Sharing signatures between IDSs would conserve valuable resources
- Each IDS has its own signature “language”, so sharing signatures is not trivial
Proposed solution

- Automated translation between signature languages
  - Simplifies signature sharing
  - Supports easier comparison of different signatures for similar attacks
    - express signatures in (or translate to) a common language
- Potential benefits of research
  - Leads to greater insight into attack language requirements
    - what can language A do that B cannot, and vice versa?
  - Has not been done before
    - ArachNIDS database supports generation of signatures for several IDSs with similar rule languages
      - Snort, Dragon, Pakemon, DefenseWorx, Shoki
Translation issues

• **Language compatibility**
  – Cannot translate features that don’t exist in target language
  – Domain-dependent factors
    • protocols (ethernet, IP, TCP, UDP, ICMP, DNS, ...)
    • protocol fields
    • user-defined functions
  – Domain-independent factors
    • multi-event patterns
      – sequence, or, and, loop, time, ...

• **Other factors**
  – Are generated signatures “as good as” hand-crafted signatures?
  – Is automated translation cost-effective?
Why snort

- Snort has a simple, concise language for expressing detection signatures
- Snort signatures are readily available
- Several other network IDSs have signature languages essentially equivalent to snort
• A snort rule is a detection signature for matching single events
  – Snort uses preprocessors to match signatures too complex for rule language
• Each rule has a rule header and rule options
  – Rule header matches “action”, IP addresses, and ports
  – Rule options match protocol fields and payload content
• Example
  
  ```
  alert tcp $EXTERNAL_NET any -> $HOME_NET 21
  (msg:"FTP passwd attempt";flags: A+; content:"passwd");
  ```
Why STATL

- Domain-independent attack language
  - Extensions for
    - IP networks (NetSTAT)
    - Solaris BSM
    - WinNT event logging facility
    - Apache event logs
    - Syslog facility
    - IDMEF alerts
- Much more expressive than snort, etc.
- Text and graphical form
- Potentially the “common language” mentioned earlier
STATL basic abstractions

- Scenario
  - States
  - Transitions (consuming, nonconsuming, unwinding)
  - Signature actions
  - Assertions
  - Global environment
  - Local environment
  - Code blocks
- Events
- Timers
NetSTAT example

use tcpip;

scenario streambin {
    string CLASSIFICATION_NAME = "Streambin";
    string CLASSIFICATION_URL = "http://www.cs.ucsb.edu/~rsg";
    string SOURCE_NODEADDRESS = "unknown";
    string SOURCE_PORT = "unknown";
    string TARGET_NODEADDRESS = "unknown";
    string TARGET_PORT = "unknown";
    string ADDITIONAL_DATA = "Binary packet: ";
    int sid;

    transition open (s0->data) nonconsuming {
        [STREAMOpen s] : s.header.type == STREAM_EVENT_OPEN_C2S &&
            (s.header.getDstPort() == 25 ||  // smtp
             s.header.getDstPort() == 21 ||  // ftp
             s.header.getDstPort() == 110)  // pop
            { sid = s.header.id;
              SOURCE_NODEADDRESS = s.header.getSrcStr();
              SOURCE_PORT = s.header.getSrcPortStr();
              TARGET_NODEADDRESS = s.header.getDstStr();
              TARGET_PORT = s.header.getDstPortStr();
            }
    }

    transition data (data->data) consuming {
        [STREAM s] : (s.header.type == STREAM_EVENT_DATA_C2S &&
            s.header.id == sid && !s.containsBinary())
            { ADDITIONAL_DATA += s.asString(); }
    }

    transition binary (data->binary) consuming {
        [STREAM s] : (s.header.type == STREAM_EVENT_DATA_C2S &&
            s.header.id == sid && s.containsBinary())
            { ADDITIONAL_DATA += s.asString(); }
    }

    transition close (data->s0) unwinding {
        [STREAMClose s] : s.header.id == sid
    }

    initial state s0 { }
    state data { }
    state binary {
        { log("Streambin compromised"); }
    }
}
NetSTAT example
Translating snort to STATL

constraints

• Detection and response are independent concepts in STAT
  – Responses are dynamically associated with signatures at runtime

• Snort reaction/response features are not translated
  – Most snort action types (e.g., alert)
  – Output options msg and logto
  – Response/reactions options resp and react

• No attempt to “translate” snort preprocessors
  – e.g., portscan
Translating Snort to STATL

Translation rules

- **Variables**
  - `$NAME` translates to scenario parameter `NAME`

- **Rule header - action**
  - Rule actions `alert` and `log` translate to nothing - not part of signature
  - Rule actions `activate` and `dynamic` translate to looping scenario
  - Rule action `pass` not translatable
    - no such semantics in STATL
Translating snort to STATL translation rules

- Rule header - protocol, IP addresses, ports
  - Protocol translates to event spec
    - *tcp* translates to \([\text{IP} \ \text{ip} \ [\text{TCP} \ \text{tcp}]]\), etc.
  - Source and destination IP addresses
    - *192.168.1.0/24* translates to
      \(\text{ip.header.srcMatch("192.168.1.0/24")}\)
    - *any* translates to nothing
  - Source and destination ports
    - *21* translates to \(\text{tcp.header.getDstPort()} == 21\)
  - Direction
    - determines only which IP address and port specify source and which specify destination
Translating snort to STATL translation rules

- Rule options
  - Most snort options translate directly to a STATL condition
    - `ttl:n` translates to `ip.header.ttl == n`
  - String matching
    - `content:string; offset:n; depth:m;` translates to
      `tcp.payloadMatch(string, n, m, nocase?)`
alert tcp $EXTERNAL_NET any -> $HOME_NET 21
   (msg:"FTP passwd attempt";flags: A+; content:"passwd");

transition t26 (s0->s1) nonconsuming {
   [IP ip [TCP tcp]] :
      ip.header.srcMatch($EXTERNAL_NET) && ip.header.dstMatch($HOME_NET)
      && (tcp.header.getDstPort() == 21)
      && (tcp.header.flags & (TH_ACK))
      && tcp.payloadMatch("passwd",0,0,false)
   {
      CLASSIFICATION_NAME = "FTP passwd attempt";
      SOURCE_NODEADDRESS = ip.header.getSrcStr();
      TARGET_NODEADDRESS = ip.header.getDstStr();
      SOURCE_PORT = tcp.header.getSrcPortStr();
      TARGET_PORT = tcp.header.getDstPortStr();
   }
}
Snort-to-STATL summary

- Most snort rules translate directly
  - Snort preprocessors may be used to implement “complex” signatures
    - preprocessors are essentially free-form
    - automated translation impractical

- Snort “pass” rules cannot be translated to STATL
  - Sensor control issue, not a signature issue

- Snort can be directed to exit on undefined variables
  - Snort runtime issue, not a signature issue

- Redundancy between TCP and UDP rules
  - Abstract signature does not depend on protocol

- “Families” of snort rules
Translating rule "families"

- One scenario per rule
- One transition per rule
- One transition per family
Translating STATL to snort constraints

- Multi-event scenarios cannot be translated to snort rules
  - Each snort rule applies to single packets
  - A scenario may have multiple transitions, but all transitions must share the initial state and the final state
  - No unwinding transitions
- Only basic event types for IP, TCP, UDP, and ICMP
  - E.g., IP_datagram is ok, IP_fragment is not
  - No other protocols
- No scenario functions
- No state variables
- No state codeblocks and limited transition codeblocks
Only a very limited subset of STATL can be translated to Snort.

- For that limited subset, translation is straightforward.

Most of the 25+ signatures in the “standard” NetSTAT scenario set cannot be translated to Snort rules.

- Multiple transitions (events), unsupported protocols, or abstract events.
- STAT encourages a relatively small number of sophisticated signatures.
- Snort encourages a relatively large number of simple signatures.
Other languages investigated

• N-code (NFR)
  – An NFR backend consists of configuration files, recorders, and N-code filters
  – Configuration files and recorders specify what data is recorded and where
  – Filters specify which events match
    • each filter has a name, a trigger type (i.e., event), trigger modifiers (like snort protocol-specific rule options), and a “codeblock”
    • analogous to STATL transitions
  – Is it possible and practical to translate between NFR backends and STATL (NetSTAT) scenarios?
N-code translation investigation

- Translated a small number of simple signatures in each direction
- Translated one substantial NFR backend - a webserver detector - from NFR to STATL
- Developed a set of rules for translating NFR backends to STATL
- Applied STATL-to-NFR rules by hand to create a new version of the webserver detector backend
  - Functionally identical to the original, but structurally very different
N-code translation summary

• N-code to STATL
  – Identify implicit states and transitions, if any
  – Everything else is as easy as snort to STATL

• STATL to N-code
  – Represent STATL states with N-code global variables
  – Use only event types that correspond to NFR triggers
  – Use only event fields that correspond to NFR trigger modifiers

• It might be practical to automatically translate in either direction, with some limitations on what can be translated
Summary of lessons learned

- Tight coupling between detection and response complicates signature development and sharing
- With respect to detection (ignoring response features)
  - STATL appears to be a superset of snort
  - STATL appears to be a superset of N-code (but the difference is less)
  - Snort rules are very concise but not very expressive
  - STATL and N-code are very expressive but not nearly as concise as snort
    - N-code is more concise than STATL for simple signatures
  - How groups of related rules/filters/transitions are translated may significantly affect performance
Conclusions and future work

- Is signature sharing practical?
  - Results so far are promising

- Is language translation useful?
  - Yes, writing signatures is a human-intensive task

- Is a common attack language feasible or useful?
  - Might allow a CVE-like (or ArachNIDS-like?) database of signatures

- More to be done
  - Other interesting languages: bro, P-BEST
  - More rigorous specifications of signature categories that can be represented in various signature languages
  - Performance studies to identify whether translations are practical
• This slide intentionally blank
Attack languages

- Event languages describe basic events for security analysis
- Response languages define actions to be taken after detection
- Report languages are used to share information about attacks
- Correlation languages specify relationships among attacks
- Exploit languages define steps to be followed to perform an intrusion
- Detection languages provide mechanisms/abstractions to identify the manifestation of an attack