Signature Based Intrusion Detection Systems

Philip Chan
CS 598 MCC
Spring 2013
Intrusion Detection Systems

Detect malicious activities/attacks
- Hacking/ unauthorized access
- DOS attacks
- Virus/ Malware

Log events
- For Forensics and security auditing

Raise alarms
- Alert administrators
- Trigger defense mechanism if available

React to attacks
- Disconnect attack channels
- Quarantine infected systems
Network IDSs

- Monitors and analyzes data packets on a network to look for suspicious activity
- Large scale servers can monitor backbone network links
- Small scale systems can monitor local routers/switches
- Two major approaches
  - Signature based (this lecture)
  - Anomaly detection based
Signature Based IDS

**Advantages**
- Simple to implement
- Lightweight
- Low false positive rate
- High true positive rate for known attacks

**Disadvantages**
- Low detection rate for zero day attacks
Signature Based IDS

Key Challenges

- Packet analysis is major bottleneck
  - How fast can signature string matching be done?
    - Exact string matching
    - Approximate string matching
**Example**

**SNORT**

- **Rule Matching**

  - {TCP, 80, "Perl.exe", ...}

- **Match?**

  - Yes

  - **Action**

- **Match?**

  - No

  - **Dropped**

*Snort is passive wiretapping*
Aho-Corasick Algorithm

- One pass multi-string matching
  - Can find all occurrences of any number of keywords in a string, in ONE pass
- Constructs a finite state string pattern machine
- Construction of state machine proportional to sum of lengths of keywords
- FSM input: text string
Aho-Corasick Algorithm

- Naive approach
  - Assume keyword starts at byte 0 of payload, traverse trie
  - If failed, start from byte 1 and traverse, etc
  - Worst case: $L \times h$
    - $L$ : length of payload
    - $h$ : height of trie
Aho-Corasick Algorithm

- Aho-Corasick
  - Computes internal failure pointers and suffix pointers
    - Eliminates needs to backtrack and restart at top of trie every time
  - Complexity: $O(\text{len(payload)} + \#\text{pattern occurrences})$
    - assuming FSM is precomputed offline
Aho-Corasick Algorithm

- Keywords: \{test, telephone, phone, elephant\}
- Solid lines: Normal transitions
- Dotted lines: Failure transitions
Aho-Corasick Algorithm

The pot had a handle.

Searched words: The, han, and, pork, port, pot, ha, e
Boyer-Moore Algorithm

- Fast one pass *single-string* matching
- Explicit character comparison at different alignments of keywords in payload
  - Keywords preprocessed
  - Skip as many alignments as possible
- Compare strings from END of keywords
- Usually very fast in practice
  - skips a large portion of characters
  - compared to Aho-Corasick which goes through whole string regardless
Boyer-Moore Algorithm

- Shifting through alignments
  - Start with last char in keyword
  - Match: continue
    - All match: word found in payload
  - Not match: does char exist in keyword?
    - Yes: shift to that char closest to current position
    - No: skip whole string
  - Continue
Boyer-Moore Algorithm

- Slide keywords along payload
- Match compare from END of keywords
  - Example
Boyer-Moore Algorithm

- Concurrent multi-keyword comparisons
  - Trunc all keywords to length of shortest keyword
  - Build trie in reverse (start from end of truncated keywords)
    - so concurrent comparison only requires current packet char to index into trie node
  - On success: continue down trie
    - If at leaf, check if truncated characters match
    - For small number of strings, this generally performs better than Aho-Corasick in implementation
  - On failure: shift by precomputed amount in failure pointer
Performance

● In practice, Aho-Corasick and Boyer-Moore provides little performance improvement
  ○ Very little packets match a large number of strings/signatures
    ■ Naive method would generally also do well
  ○ More overhead due to code complexity

● However, large improvement for \textit{worse-cast} traces
  ○ May be crucial from hardware perspective

● A lot of research in effort to enhance Aho-Corasick/Boyer-Moore to further improve performance
Snort

Source: Nalneesh Gaur, Snort: Planning IDS for your enterprise
Snort

Figure 1-5 Components of Snort.

Source: Rafeeq Ur Rehman, *Intrusion Detection Systems with Snort: Advanced IDS Techniques with Snort, Apache, MySQL, PHP, and ACID*
Snort - Detection Engine

- Rule Pattern Searching
- Boyer-Moore

Boyer-Moore works most efficiently when the search pattern consists of non-repeating sets of unique bytes. For example, in x86, avoid including 0x90 (NOP) in the pattern to avoid repeated partial matches.
Snort - Rules

- written in single line in snort config file
- created by known signatures
- rule (type) scanning order
  - Alert -> pass -> log

Source: Nalneesh Gaur, Snort: Planning IDS for your enterprise
Questions?