Robust and Fast Pattern Matching For Intrusion Detection

Kedar Namjoshi and Girija Narlikar
Bell Labs
INFOCOM 2010
Intrusion Detection in a picture

- BAD PACKET SPECIFICATION
  - PACKET
    - NO MATCH (OK)
    - MATCH (BAD)
Intrusion Detection in more detail

- Stage 0: IP address/flow-based filter
- Stage 1: Keyword match (Aho-Corasick algorithm)
- Stage 2: **Regex match**
An unexpected performance glitch

Snort slows down to **ONE** packet/second!

The slowdown is in the regex filter.

Why?
Regular Expressions and Non-Deterministic Finite Automata (NFAs)

Regular expression syntax

\[ R := \epsilon \mid a \ (a \in \Sigma) \mid R; R \mid R + R \mid R^* \]

Thompson (1968) gives a linear-time translation to NFAs by induction on syntax. E.g., NFA for \( R^* \) is

Examples

- \((a^*)^*\) ... a sequence of a's
- \(((a+b)^*a^x)(a+b)^*\) ... 100 a's somewhere in an \{a,b\} string
- \((a?)^n a^n\) ... optionally \(n\) a's followed by \(n\) a's
Pattern Matching Algorithms

Regular Expression

<table>
<thead>
<tr>
<th>NFA</th>
<th>Convert to</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DFA</td>
</tr>
<tr>
<td></td>
<td>Table lookup (lex)</td>
</tr>
</tbody>
</table>

Backtracking algorithm (pcre)

On-the-fly determinization [Thompson 1968] (grep)
Properties

- Regular expression $\rightarrow$ NFA: linear
- NFA $\rightarrow$ DFA: exponential space in worst case
- Backtracking: exponential time in worst case
- Thompson's algorithm (on-the-fly): $|NFA| \times |\text{packet}|$ worst-case, $|NFA|+|\text{packet}|$ in practice if memoized (Dragon Book)

Why use backtracking at all?

1. When packet matches expression, may do less work than Thompson's algorithm
2. Easy to adapt to back-references
Back-References

Express *non-regular* properties: \((.*)\1\) matches words of the form \(ww\)

Matches “abab” and “aa” but not “aaa” or “abba”

\1 is a back-reference whose match must be identical to the substring captured by the bracketed expression 

Increasingly important for IDS. Data for Snort:
The Problem in a Nutshell

- Thompson's on-the-fly algorithm has linear complexity for regular expressions
- Back-reference matching uses backtracking
- **Problem:** a specification with back-references mixed up with bad regular expressions
  > potentially exponential matching complexity
  > IDS open to a low-frequency DOS attack
- **Q:** Can one detect the potential for exponential behavior off-line?
  > NP-hard
- **Our solution:** Extend Thompson's on-the-fly determinization algorithm to back-references
On-The-Fly matching for Regex's with Back-references

- Thompson's algorithm records the set of active NFA states which may be reached after reading a prefix of the input packet.

- Our algorithm is similar, except that it records a set of configurations. A configuration is a triple (q, map, state) where:
  - "q" is an NFA state
  - "map" is a (possibly partial) mapping of brackets to a substring of the packet
  - "state" indicates whether it is necessary to match a back-reference at q

- The set of configurations is updated when:
  - A new bracket is "opened" or "closed" - e.g., at "(“ and "")" in (.*)\1
  - A new input byte is read from the packet

- Worst-case space complexity = \# of possible configurations = |NFA| \* |packet|^{2K}, where K = \# back-references

- Worst-case time complexity = |space| \* |packet|
Refining worst-case bounds

- The worst-case bounds are rather loose. For instance
  - \((ab)\1 + (cd)\2\) has two backrefs, but only one is used on any run
  - In \((ab)(cd)\1(ef)\2\), the first backref is not used anywhere after the first occurrence
  - In \((ab)(R)\1\), the backref can only be bound to “ab”
- We detect such cases with “liveness analysis” of the backref-NFA (a technique originally developed in compiler optimization)

- On Snort rules
  - 120 distinct backref expressions; backref-NFA's have between 70 and 30,000 states
  - 111 expressions have a single backreference
  - 8 expressions have 6 backreferences but at most 2 live at any state
  - 1 expression has 4 live backreferences but each has a fixed match, so the true space complexity is linear and not \(|NFA|^*|packet|^2\)
- So the worst-case bounds are usually not reached in practice
- Liveness information can be further used to optimize the matching algorithm by restricting configurations to live references
Experiments with Snort (I)
Experiments with Snort (II)

- Backtracking does less work because the traces are “bad” traces (i.e., they match the specification)

- For “good” traces, the work done by the deterministic algorithm is at most that required for backtracking
Closely Related Work

- K. Thompson, “Regular Expression Search Algorithm” [C.ACM 1968]
- S. Crosby, “Denial of service through regular expressions” [USENIX 2003]
- M. Becchi and P. Crowley, “Extending finite automata to efficiently match Perl-compatible regular expressions” [CONEXT 2008]
DPI and Model Checking

- DPI is verification on a finite path!
- Expressiveness/compactness vs. model checking complexity trade-off for specification languages
- Temporal logic can be checked in linear time on a finite path (folklore)
- DPI could exploit such tradeoffs, both for flexibility and for robustness
To conclude...

- Unrestricted use of regular expressions (with or without back-references) can result in exponential matching complexity
- This is a serious and real robustness problem
- Deterministic algorithms provide robust guarantees
- Automaton analysis can help detect bad cases at compile time
- There is much more that could be done to analyze NFA structure
- There is much more that could be done with specification languages, to exploit the trade-off between expressiveness/compactness vs. matching complexity