Curing Regular Expressions
Matching Algorithms from
Insomnia, Amnesia, and Acalculia

Sailesh Kumar
Balakrishnan Chandrasekaran
Jonathan Turner
George Varghese
Regular Expressions in Security

- Signature based NIDS is a popular device to enable network security.

- Attack patterns are specified as regular expressions.
  - \[ \ \t\]*[Cc][Ww][Dd]\[ \ \t\]+[~]root
    - Represents an attempt to change working directory to root.

- Regular expression matching is expensive.
  - Thousands of signatures.
  - High speed implementation requires GB memory. (often impractical.)
Traditional Implementation

- NIDS implementation.

Traditional implementation attempts to match traffic with the entire Virus signature.

Complex signatures lead to trade-off.

<table>
<thead>
<tr>
<th>DFA: Fast, but requires large memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFA: Compact, but slow</td>
</tr>
<tr>
<td>D²FA: Trades-off memory-performance</td>
</tr>
</tbody>
</table>

Signatures:
- \( r1 = .*[gh]d[^g]*ge \)
- \( r2 = .*fag[^i]*[^j]*j \)
- \( r3 = .*a[gh]i[^l]*[ae]c \)
**Insomnia**

- NIDS implementation.

**OBSERVATION:**

Typical traffic rarely match first few symbols within any virus signature.

NIDS keeps the entire signature active.

(Unvisited tail portions can be kept to sleep)

We refer to this problem as Insomnia
Cure to Insomnia

• Solve Insomnia with a three-way trade-off.

- Smaller matching signature prefixes => high performance, low memory
- In practice, frequently matching prefixes are very small in length

Diagram:
- 1/ Memory -> NFA
- NFA -> D²FA, etc
- Performance
- Traffic characteristics
- Memory

Graph:
- DFA
- D²FA, etc
Cure to Insomnia

- Insomnia cure.

- If we select prefixes, then:
  - Prefixes are small
  - Few packets match them – goto slow path

  Only prefixes of signatures are matched in fast path

  Fast prefix implementation (e.g. DFA) will require less memory, and will be feasible.

  Suffixes won’t require fast implementation, will use less memory, and will be feasible.

  High performance, Less memory

  Suffixes of the prefix matching signatures are matched in slow path.

  How to select the prefixes?

Patterns:

- \([gh]*d[^g]+ge\)
- \(fag[^i]+[^j]+j\)
- \(a[gh][^l]+[ae]c\)
Prefix Generation

Construct the NFA
Run NFA for an input trace
Count # times state is active
Find probability of state activity
MAKE A CUT (Limit the total slow path state probability)
DoS Attacks

Attacker sends traffic that matches prefix “too often” Overloads the slow path

well behaving flows will suffer

Use per-flow anomaly counter
Counts # of packets sent to the slow path.

Flows with high anomaly counter value are attack flows
Send then to a low priority queue
Simulation of DoS Mitigation

<table>
<thead>
<tr>
<th>Time (seconds)</th>
<th>Slow Path Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-26</td>
<td>No overload</td>
</tr>
<tr>
<td>76-151</td>
<td>Moderate overload</td>
</tr>
<tr>
<td>176-251</td>
<td>Extreme overload</td>
</tr>
</tbody>
</table>

Slow path load's $\varepsilon$ threshold

Throughput with no DoS mitigation:

- Good flows
- 50 well behaving flows
- 10 become anomalous
- 20 become anomalous

Throughput with DoS mitigation:

- 50 well behaving flows
- 10 become anomalous
- 20 become anomalous

Flow throughput, DoS protection
## Results of Splitting Prefix/Suffix

<table>
<thead>
<tr>
<th>Source</th>
<th># of Rules</th>
<th>Regular expressions before split</th>
<th></th>
<th>Prefixes after split</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ASCII length</td>
<td>Number of DFA</td>
<td>Total memory</td>
<td>ASCII length</td>
</tr>
<tr>
<td>Cisco</td>
<td>68</td>
<td>44.1</td>
<td>6</td>
<td>973 MB</td>
<td>19.8</td>
</tr>
<tr>
<td>Linux</td>
<td>70</td>
<td>67.2</td>
<td>4</td>
<td>30.7 MB</td>
<td>21.4</td>
</tr>
<tr>
<td>Bro</td>
<td>648</td>
<td>23.64</td>
<td>1</td>
<td>3.77 MB</td>
<td>16.1</td>
</tr>
<tr>
<td>Snort rule 1</td>
<td>22</td>
<td>59.4</td>
<td>5</td>
<td>114.6 MB</td>
<td>36.9</td>
</tr>
<tr>
<td>Snort rule 2</td>
<td>10</td>
<td>43.72</td>
<td>2</td>
<td>64.2 MB</td>
<td>16</td>
</tr>
<tr>
<td>Snort rule 3</td>
<td>19</td>
<td>30.72</td>
<td>N/A</td>
<td>N/A</td>
<td>13.8</td>
</tr>
</tbody>
</table>

Slow path probability set to less than 0.01%
NFAs are compact but slow
  » Multiple active state

DFAs are fast representation
  » State explosion is serious problem
  » State explosion mainly occurs due to the presence of closures

Three patterns
  » 3 separate DFAs create 12 states
    - 3 active states
  » NFA has only 9 states
    - Up to 6 active state
  » A single DFA creates 20 states
    - 1 active state

(ab.*c) | (ac.*b) | (ba.*a)
State Explosion in DFA

- State explosion occurs primarily because
  - DFA has single active state
  - Don’t remember anything but the current active state (amnesia)
- Requires a separate DFA state for every situation that may occur during NFA parse

\[ (ab.\ast z) \mid (cd.\ast z) \mid (ef.\ast z) \]

**Active states**

- Input: abcd  \( \{0, 2, 5\} \)
- Input: abef  \( \{0, 2, 8\} \)
- Input: cdef  \( \{0, 5, 8\} \)
- Input: abcdef  \( \{0, 2, 5, 8\} \)

\( k \) closures => Number of DFA states is exponential in \( k \)
Our solution is History based Finite Automata (HFA)

- Enable a single state of execution
- Use a bit to represent the condition that a closure is reached
- Certain transitions depend upon the bit values
- Bits are also updated as HFA makes its transitions

\[(ab.*z) | (cd.*z) | (ef.*z)\]
Benefits of HFA

- Single State of Execution – high performance
- Few bits are required (16, 32) – stored in registers
- Avoids state explosion – memory efficient

\[(ab.*z) | (cd.*z) | (ef.*z)\]
## Results

<table>
<thead>
<tr>
<th>Source</th>
<th># of closures</th>
<th>DFA</th>
<th>H-FA</th>
<th>% space reduction with H-FA</th>
<th>H-FA parsing rate speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td># of automata</td>
<td>total # of states</td>
<td># of automata</td>
<td># of flags</td>
</tr>
<tr>
<td>Cisco64</td>
<td>14</td>
<td>1</td>
<td>132784</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Cisco64</td>
<td>14</td>
<td>1</td>
<td>132784</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>Cisco68</td>
<td>19</td>
<td>1</td>
<td>328664</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>Snort 1</td>
<td>6</td>
<td>3</td>
<td>62589</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Snort 2</td>
<td>1</td>
<td>1</td>
<td>12703</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Snort 3</td>
<td>5</td>
<td>2</td>
<td>4737</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Linux70</td>
<td>11</td>
<td>2</td>
<td>20662</td>
<td>1</td>
<td>9</td>
</tr>
</tbody>
</table>
- Thank you and Questions???