Efficient Signature Matching with Multiple Alphabet Compression Tables

Shijin Kong
Randy Smith
Cristian Estan

Presented at SecureComm 2008, Istanbul, Turkey
Signature Matching

- Signature Matching a core component of network devices

- Operation (ideal): For a set of signatures, match all relevant sigs in a single pass over payload

- Many constraints
  - Evolving, complex signatures
  - Wirespeed operation
  - Limited memory
  - Active adversary
Regular Expressions and DFAs

- Regular expressions standard for *writing* sigs
  - Buffer overflow: `/^RETR\s[^\n]{100}/`
  - Format string attack: `/^SITE\s+EXEC[^\n]*%[^\n]*%/`

- DFAs used for *matching* to input
# DFA Operation

### DFA States

<table>
<thead>
<tr>
<th>State 0</th>
<th>State 1</th>
<th>State 2</th>
<th>State 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>12</td>
<td>12</td>
<td>25</td>
</tr>
<tr>
<td>25</td>
<td>12</td>
<td>12</td>
<td>25</td>
</tr>
<tr>
<td>25</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>41</td>
<td>2</td>
<td>4</td>
<td>41</td>
</tr>
<tr>
<td>41</td>
<td>8</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>8</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

- **crt_state=1**
- **input_byte=1**

### Next State Transition

- **next_state=12**

### Alert

- `if accept(next_state)`
- `alert`
Matching with DFAs

- **Advantages**
  - Fast – minimal per-byte processing
  - Composable – combine many DFAs into one

- **Disadvantages**
  - States are heavyweight (1 KB each!)
  - State-space explosion occurs when DFAs combined

- Memory exhausted with only a few DFAs!
- Workaround: many DFAs matched in parallel
**Key: Reduce memory usage**

Strategy: aggressively reduce memory footprint, keep exec time low

- Reduce size of transition tables
- Reduce number of states
Main Contribution

- Multiple Alphabet Compression Tables
  - Lightweight, applicable to hardware or software
  - Compatible with other techniques
  - Worst case = average case

- Results (in software)
  - 4x to 70x memory reduction
  - 35% - 85% execution time increase
Outline

- Introduction
- Alphabet Compression Tables
- Interacting with D²FAs
- Experimental Results
Alphabet Compression: core observation

Some input symbols are equivalent; the transitions on those symbols at any state are identical.

<table>
<thead>
<tr>
<th>State 0</th>
<th>State 1</th>
<th>State 2</th>
<th>State 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>12</td>
<td>12</td>
<td>25</td>
</tr>
<tr>
<td>25</td>
<td>12</td>
<td>12</td>
<td>25</td>
</tr>
<tr>
<td>25</td>
<td>12</td>
<td>12</td>
<td>25</td>
</tr>
<tr>
<td>41</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>41</td>
<td>2</td>
<td>4</td>
<td>41</td>
</tr>
<tr>
<td>41</td>
<td>8</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>8</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>
Alphabet Compression Tables

Alphabet compression table

State 0
- input_byte=1
- index=0
- next_state=12
- crt_state=1

State 1
- next_state=12

State 2
- next_state=25

State 3
- next_state=25
Even further compression...

<table>
<thead>
<tr>
<th>Alphabet compression table</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>State 0</td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td>41</td>
</tr>
<tr>
<td>41</td>
</tr>
<tr>
<td>41</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Even further compression...

Alphabet compression table

<table>
<thead>
<tr>
<th>State 0</th>
<th>State 1</th>
<th>State 2</th>
<th>State 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>12</td>
<td>12</td>
<td>25</td>
</tr>
<tr>
<td>41</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>41</td>
<td>2</td>
<td>4</td>
<td>41</td>
</tr>
<tr>
<td>41</td>
<td>8</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>8</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

...
Even further compression...

Alphabet compression table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>State 0</th>
<th>State 1</th>
<th>State 2</th>
<th>State 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>25</td>
<td>12</td>
<td>12</td>
<td>25</td>
</tr>
<tr>
<td>0</td>
<td>41</td>
<td>4</td>
<td>4</td>
<td>41</td>
</tr>
<tr>
<td>0</td>
<td>41</td>
<td>2</td>
<td>4</td>
<td>41</td>
</tr>
<tr>
<td>1</td>
<td>41</td>
<td>8</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>41</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td></td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Multiple ACTs

<table>
<thead>
<tr>
<th>ACT 0</th>
<th>ACT 1</th>
<th>State 0</th>
<th>State 1</th>
<th>State 2</th>
<th>State 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>25</td>
<td>12</td>
<td>12</td>
<td>25</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>41</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td>41</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How do we know which ACT to use with which state?
Multiple ACTs

ACT 0
0 0 0 1 1 12 2 2 3

ACT 1
0 0 0 1 1 2 3 3

State 0
0 25 1 41 0 5

State 1
0 12 1 4 0 2 0 8

State 2
0 12 1 6

State 3
0 25 1 41 0 5

index=0

input_byte=1
crt_act=1
crt_state=1

next_act=0
next_state=12
Constructing Multiple ACTs

- Partition states appropriately
  - for example:
    \[
    \{S_1, S_2, S_3, \ldots, S_n\} \rightarrow \{\{S_1, S_8\}, \{S_2, S_3, S_9\}, \ldots\}
    \]

- Construct single ACT for each group of states
  - See algorithm in paper
Partitioning States for ACTs

- Input: number of ACTs to use \( m \), DFA \( D \)
- Output: a partition of states into \( m \) subsets

Use greedy, heuristic approach:

\[
\begin{align*}
\text{States} & = \text{Set of all states in } D; \\
\text{while } (m > 1) \{ \\
\quad \text{Subset} & = \text{GetEquivClassPartition(States)}; \\
\quad \text{AddToResult(Subset)}; \\
\quad \text{States} & = \text{States} - \text{Subset}; \\
\quad m & --; \\
\} \\
\text{return Result;}
\end{align*}
\]
How many Compression Tables?

- Eight ACTs is enough
Outline

- Introduction
- Alphabet Compression Tables
- Interacting with D²FAs
- Experimental Results
ACTs and $D^2$FAs

- Two kinds of redundancy

Symbols have identical behavior for large subsets of states

*Compress with (multiple) ACTs*
ACTs and $D^2$FAs

- Two kinds of redundancy

Symbols have identical behavior for large subsets of states

*Compress with (multiple) ACTs*

Symbols at many states lead to common next states

*Compress with $D^2$FAs*
D²FAs: core observation

For many pairs of states, the transitions for most characters are identical!

Idea: store only one copy

Kumar et al (Sigcomm 2006); Kumar et al (ANCS 2006); Becchi et al (ANCS 2007)


**D²FAs: core observation**

For many pairs of states, the transitions for most characters are identical!

Idea: store only one copy

Kumar *et al* (Sigcomm 2006); Kumar *et al* (ANCS 2006); Becchi *et al* (ANCS 2007)
D²FAs

Default transitions

Issue:
good compression,
potentially heavy runtime cost

input_byte=1
crt_state=1

next_state=12
ACTs and D²FAs Together

- Combine ACTs and D²FAs to address both kinds of redundancy

- Procedure:
  1. Apply D²FA compression to DFAs
  2. Apply multiple ACT compression to D²FA results

- Only slight modification to ACT construction
  - Add “not handled here” symbol
  - Deal with default transitions
ACTs + D²FAs

Default transitions

State 0  State 1  State 2  State 3

...
### ACTs + D²FAs

#### ACT 0

<table>
<thead>
<tr>
<th>State</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### State 0

- 2

#### State 1

- 2

#### State 2

- 0

#### State 3

- 0

...
ACTs + $D^2$FAs

<table>
<thead>
<tr>
<th>ACT 0</th>
<th>ACT 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

State 0 | State 1 | State 2 | State 3
--------|---------|---------|---------
0 25    | 1 41    | 0 5     | 1 6     |
0 5     | 0 2     | 0 8     | 0 5     |
0 8     | 0 2     | 1 4     | 1 41    |
1 0     | 1 0     | 1 0     | ...     |
ACTs + $D^2$FAs
Outline

- Introduction
- Alphabet Compression Tables
- Interacting with $D^2$FAs
- Experimental Results
Experimental Setup

- 1550 HTTP, SMTP, FTP signatures
  - Grouped by protocol and rule set (Snort or Cisco)

- DFA Set Splitting (Yu, 2006) to cluster DFAs
  - Provide memory bound \textit{a priori}
  - Heuristically combine into as few DFAs as possible

- Experiment Environment
  - 10 GB traces, run on 3.0 GHz P4
  - Exec time measured with cycle-accurate counters
Memory vs Time

![Graph showing Memory vs Time trade-off for DFAs with different state counts. The x-axis represents Memory usage (KB), and the y-axis represents Execution Time (cycles/byte). The graph includes points for 4000 States with 114 DFAs, 8000 States with 82 DFAs, and 16000 States with 45 DFAs. The ideal point is marked with an 'ideal' label.](image)
Memory vs Time

Snort HTTP

Cisco IPS HTTP

Lowest mem, highest exec!
Memory vs Time

Snort SMTP

Cisco IPS SMTP

Lowest mem, highest exec!
Conclusion

- Multiple alphabet compression tables
  - Lightweight
  - Applicable to hardware or software platforms
  - Compatible with other techniques

- Provides better time vs. space performance
  - 4x to 70x memory reduction
  - 35% to 85% execution time increase

- Best technique a function of time, memory limits
  - ACTs add superior design points
Efficient Signature Matching with Multiple Alphabet Compression Tables

Thank you
intentionally blank
Regular Expressions and DFAs

- Regular expressions standard for *writing* sigs
  - Buffer overflow: `/^RETR\s[\^\n]{100}/`
  - Format string attack: `/^SITE\s+EXEC[\^\n]*%[\^\n]*/`

- DFAs used for *matching* to input

---

```
C0 A8 64 01  site exec % retr %
```

---

Match sig 2!
## Memory Usage

<table>
<thead>
<tr>
<th></th>
<th>DFA</th>
<th>ACT</th>
<th>D²FA</th>
<th>ACT + D²FA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Snort HTTP</strong></td>
<td>74</td>
<td>8.1</td>
<td>8.8</td>
<td>4.3</td>
</tr>
<tr>
<td><strong>Snort SMTP</strong></td>
<td>98</td>
<td>5.7</td>
<td>42</td>
<td>3.4</td>
</tr>
<tr>
<td><strong>Snort FTP</strong></td>
<td>94</td>
<td>4.9</td>
<td>9.2</td>
<td>3.9</td>
</tr>
<tr>
<td><strong>Cisco HTTP</strong></td>
<td>116</td>
<td>30</td>
<td>4.7</td>
<td>17</td>
</tr>
<tr>
<td><strong>Cisco SMTP</strong></td>
<td>110</td>
<td>29</td>
<td>3.0</td>
<td>18</td>
</tr>
<tr>
<td><strong>Cisco FTP</strong></td>
<td>83</td>
<td>5.1</td>
<td>1.7</td>
<td>1.9</td>
</tr>
</tbody>
</table>

All results reported in megabytes (MB)
Memory vs Time

Snort FTP

Cisco IPS FTP