Intrusion Detection and Malware Analysis

Automatic signature generation

Pavel Laskov
Wilhelm Schickard Institute for Computer Science
The quest for attack signatures

- Post-mortem: security research, computer forensics
- Reactive: analysis of anomalies (forensic sinks)
- Proactive: acquisition and analysis of malicious data
A general framework for ASG

- **Clustering**: finding groups of similar malicious events
- **Token extraction**: finding common patterns in malicious data
- **Signature assembly**: assessment of extracted tokens
A set of tokens $t_1, \ldots t_n$
A set of support values $\nu_1, \ldots, \nu_n$
A threshold $\theta$
Evaluation rule:

$$\sum_{i=1}^{n} \nu_i M(t_i, s) > \theta,$$

where

$$M(t_i, s) = \begin{cases} 
1 & \text{if } t_i \text{ is present in a string } s \\
0 & \text{otherwise}
\end{cases}$$
Invariance as a main principle of ASG

- Invariance is inherent for attacks due to extremely specific nature of exploits.
- Diversity makes signatures more general and accurate.
- Too much diversity makes signatures smaller and leads to false positives.
• A token is a substring found in malicious content that satisfies pre-defined empirical conditions, such as:
  • minimal length
  • minimal support: percentage of malicious events it occurs in

• A pair of tokens is said to be distinct if they are not a substring of one another.

• A token $s$ that is a substring of another token $t$ is ignored unless it satisfies tokenization conditions while being not part of $t$. 
**Problem**

Given a set \( \{s_1, \ldots, s_n\} \) of malicious payloads, find a set \( \{t_1, \ldots, t_k\} \) of tokens, such that \( |t_i| > L_{\text{min}} \) and each \( t_i \) occurs in at least \( \nu \% \) of malicious payloads.

**Remark**

Unlike many other applications of tokenization, tokens for ASG are not defined in terms of delimiters. Such delimiters may not be known in advance.
Token extraction using GST

- Traverse a GST from top to bottom.
- For each node, output its path from the root if its depth is greater than $L_{\text{min}}$ and the number of non-zero entries in its leaf count is greater than $\nu n$.
- Output the percentage of non-zero entries in its leaf count as a token support.
- Input: strings “abbaa” and “baaaa”, $L_{\text{min}} = 1$, $\nu = 100$
- Output:
- Input: strings “abbaa” and “baaaa”, $L_{\text{min}} = 1$, $\nu = 100$
- Output: “a”
Token extraction example

- Input: strings “abbaa” and “baaaa”, $L_{\text{min}} = 1$, $\nu = 100$
- Output: “a”, “aa”
Input: strings “abbaa” and “baaaa”, $L_{\min} = 1$, $\nu = 100$
Output: “a”, “aa”, “b”
Token extraction example

- Input: strings “abbaa” and “baaaa”, $L_{\text{min}} = 1$, $\nu = 100\%$
- Output: “a”, “aa”, “b”, “ba”, “baa”
Open problems

- How can we define unique “end-of-string” markers for a full alphabet of byte values?
Open problems

- How can we define unique “end-of-string” markers for a full alphabet of byte values?
  - Escaping
  - Special encoding: extra bytes (bits)
Open problems

- How can we define unique “end-of-string” markers for a full alphabet of byte values?
  - Escaping
  - Special encoding: extra bytes (bits)
- How can we avoid generation of non-distinct tokens?
Open problems

- How can we define unique “end-of-string” markers for a full alphabet of byte values?
  - Escaping
  - Special encoding: extra bytes (bits)

- How can we avoid generation of non-distinct tokens?
  - Post-processing
  - Complex suffix tree tricks
- **Goal**: remove tokens that frequently occur in normal traffic.
- **Rules for removal**:
  - $v_-(t_i) > v_+(t_i)$
  - $v_-(t_i) > v_{\text{max}}$
- **Underlying problem**: set matching.
- **Algorithms**:
  - Knuth-Morris-Pratt: $O(k(n + M))$
  - Aho-Corasick: $O(k + n + M)$
Given the set of token/support pairs \( \{(t_1, \nu_1), \ldots, (t_k, \nu_k)\} \), signature refinement consists of the following steps:

- **Normalization**: support values are normalized so that they add up to 1:
  \[
  \nu_i = \frac{\nu_i}{\sum_{j=1}^{k} \nu_k}
  \]

- **Calibration**: the threshold \( \theta \) is calibrated on benign data so as not to exceed some maximal false positive rate.
Lessons learned

- Automatic signature generation enables one to quickly extract signatures for samples of malicious and benign traffic.
- Careful choice of algorithms and data structure is important for practical feasibility of ASG.
- ASG enable some very interesting applications to malware analysis, especially detection of malware communication.
Recommended reading

D. Gusfield.  
*Algorithms on strings, trees, and sequences.*  

Konrad Rieck, Guido Schwenk, Tobias Limmer, Thorsten Holz, and Pavel Laskov.  
Botzilla: Detecting the "phoning home" of malicious software.  
(to appear).