Polygraph: Automatically Generating Signatures for Polymorphic Worms

James Newsome Brad Karp Dawn Song

Presented by:

Jeffrey Kirby
Overview

- Motivation
- Polygraph Signature Generation Algorithm
- Evaluation
- Attack Analysis
- Related Work
- Conclusion
Motivation

Worms

- “A computer worm is a software program that is designed to copy itself from one computer to another, without human interaction. Unlike a computer virus, a worm can copy itself automatically.”[2]

- Some popular examples:
  - ILOVEYOU (2000)
  - Code Red (2001)
  - Sasser (2004)
  - Conficker (2008)
  - Stuxnet (2010)
Motivation

Mitigation Strategies

- Better, more secure software
  - Alter source code (commercial software?)
  - Defend against *every* threat

- Security updates, patches
  - After-the-fact

- Intrusion Detection Systems (IDSs)
  - Prevent the exploit from accessing the system
Motivation

Intrusion Detection Systems (IDSs)

- Monitors network for suspicious activity or traffic
- May send an alert when such activity or traffic is found
- May block traffic from source IP address
- May be network-based (NIDS) or host-based (HIDS)
- May be signature-based (SBS) or anomaly-based (ABS)
Motivation

Signature-based Intrusion Detection Systems (SBSs)

- Compares flows on the network against a database of signatures corresponding to known attacks

- A signature may be some portion of a worm's payload which is long enough to correctly match the worm without causing false positives

- Signatures are created by security experts who study network traces after an attack has taken place (hours or days)

- IDS only as good as the signatures used for matching
Motivation

Signature-based Intrusion Detection Systems (SBSs)

• In order to react more quickly to attacks, *automated* signature generation has been developed

• Pattern-based
  • Honeycomb
  • Autograph
  • Earlybird
  • Polygraph

• Semantic-based
  • TaintCheck
  • *motif-finding*
Motivation

Signature-based Intrusion Detection Systems (SBSs)

• Whether manual or automated signature generation, both assume there is a unique contiguous substring which will remain invariant across worm connections and is long enough to be specific

• But, a worm may be polymorphic, meaning that its content can change among different instances while still having the same effect

• In this case, a single contiguous substring will not match all variants of a given worm

• Polymorphic libraries are readily available
Polygraph

Three classes of bytes

1. Invariant bytes
   • must remain the same in order for the exploit to function properly
     (e.g. frames / overwrite values)

2. Code bytes
   • encrypted and/or obfuscated code which will be executed by the
     worm

1. Wildcard bytes
   • values do not affect exploit functionality
Polygraph

Three Signature Classes for Polymorphic Worms

1. Conjunction signatures
   • matches if all tokens in the set are found in the payload

2. Token-subsequence signatures
   • similar to conjunction but with order constraint
   • therefore, more specific than conjunction

3. Bayes signatures
   • tokens present in the flow used to calculate a probability that the flow is a worm
   • considered a worm if the score is above some threshold
Polygraph

Signature Generation

1. Preprocessing
   • This process separates the suspicious flows into tokens
   • A token is a contiguous byte sequence

2. Single Signature Generation
   • Conjunction
   • Token-subsequence
   • Bayes

3. Multiple Signature Generation
Polygraph

Preprocessing

• extract all distinct substrings with minimum length $\alpha$ which occur in at least $K$ out of the total $n$ samples

• distinct means that a substring should not be considered a token if it is a substring of another token, unless it is found in at least $K$ out of $n$ samples not as a substring of the larger token

• Example: if “HTTP” is a token (occurs in at least $K$ out of $n$ samples), “TTP” is only considered a token if it is found in at least $K$ out of $n$ samples not as a substring of “HTTP”
Polygraph

Preprocessing (cont'd.)

• uses a modification of *longest substring algorithm* to return a set of substrings which include all distinct substrings

• non-distinct substrings may be *pruned* to discard irrelevant, non-token data

• from this point on, each flow is represented as a sequence of tokens only

• this process runs in linear time with respect to the number of samples
Polygraph

Single Signature Generation (Conjunction)

• Conjunction signature is an unordered set of tokens

• This is trivially found by taking the union of all tokens from each sample in the flow pool
Polygraph

Single Signature Generation (Token-subsequence)

- Token-subsequence is an ordered list of tokens
- Need to find a token-subsequence which is present in *every* sample in the flow pool
- These token subsequences may be represented as regular expressions
- Example: Comparing two strings

```
<table>
<thead>
<tr>
<th>o</th>
<th>x</th>
<th>n</th>
<th>x</th>
<th>e</th>
<th>x</th>
<th>z</th>
<th>x</th>
<th>t</th>
<th>w</th>
<th>o</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>t</td>
<td>w</td>
<td>o</td>
<td>y</td>
<td>o</td>
<td>y</td>
<td>n</td>
<td>y</td>
<td>e</td>
<td>y</td>
<td>z</td>
</tr>
</tbody>
</table>
```

Which is better?

```
.*o.*n.*e.*z.*

.*two.*
```
Polygraph

Single Signature Generation (Token-subsequence)

• Use Smith-Waterman algorithm to find “best” subsequence (best meaning that it has a smaller chance of producing false positives)

• Each subsequence is given a score:
  • 1 for a match
  • Subtract $W_g$ for every gap (.* ) not including leading and trailing gaps ($W_g = 0.8$)
  • Higher score means better subsequence

  .*o.*n.*e.*z.* = 4 - 3(0.8) = 1.6
  .*two.* = 3 - 0(0.8) = 3

• The algorithm is applied iteratively to the entire pool of samples
Polygraph

Single Signature Generation (Bayes)

- Bayes provides a probabilistic matching

- Given probability distributions for sets of tokens corresponding to worm flows and innocuous flows, a sample flow can be classified as either a worm or innocuous by calculating which distribution its token set is more likely to have been generated by.

\[
P(W \rightarrow S) = x \\
P(I \rightarrow S) = y
\]
Polygraph

Multiple Signature Generation

• Need to generate set of signatures given a suspicious flow pool which may contain multiple worms as well as innocuous flows (noise)

• The Bayes algorithm needs no modification for this situation

• Conjunction and token subsequence require clustering

• Clustering divides the suspicious pool into clusters, each of which contains similar flows

• A signature is generated for each cluster
Polygraph

Multiple Signature Generation (Clustering)

• Cluster quality:
  • Should not be too general (high false positive rate)
  • Should not be too specific (high false negative rate)
Polygraph

Multiple Signature Generation (Clustering)

- Hierarchical Clustering method:
  - Start with $s$ flows and $s$ clusters, each with its own (very specific) signature
  -Iteratively merge clusters and generate new (less specific) signatures for each cluster
  -Choose the two clusters to merge by calculating which pair gives the lowest false positive rate when compared against the innocuous pool
  -Merging stops when only one cluster remains
  -OR no two clusters may be merged without resulting in a signature which gives an unacceptably high false positive rate
  -The signatures from each cluster(with enough samples) form the set of signatures
Evaluation

Experimental Setup

• Evaluate each signature generation algorithm where the suspicious flow pool contains:
  1. One worm / No innocuous flows
  2. One worm / Some innocuous flows
  3. Multiple worms / Some innocuous flows

• Token-extraction threshold $K = 3$
• Minimum token length $\alpha = 2$
• Minimum cluster size is 3
• 5 independent trials for each experiment
Evaluation

Experimental Setup

- Generate signatures for polymorphic versions of three real-world exploits:
  1. Apache-Knacker (HTTP)
  2. ATPhttpd (HTTP)
  3. BIND-TSIG (DNS)

- Simulate *ideal polymorphic engine* by filling wildcard and code bytes with randomly chosen data
Evaluation

Experimental Setup

- HTTP experiments
  - 5-day network trace (45,111 flows) as innocuous pool
  - 10-day trace (125,301) as evaluation trace
  - Evaluation trace used to measure false positive rate of signatures
  - Also, noise was drawn at random from the evaluation trace

- DNS experiments
  - 24 hour DNS trace
  - First 500,000 flows as innocuous
  - Last 1,000,000 flows as evaluation trace
Evaluation

Experiment 1. One worm / No innocuous flows

<table>
<thead>
<tr>
<th>Class</th>
<th>False +</th>
<th>False -</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Longest Substring</td>
<td>92.5%</td>
<td>0%</td>
<td>HTTP/1.1 \r\n</td>
</tr>
<tr>
<td>Best Substring</td>
<td>.008%</td>
<td>0%</td>
<td>\xFF\xBF</td>
</tr>
<tr>
<td>Conjunction</td>
<td>.0024%</td>
<td>0%</td>
<td>‘GET’, ‘HTTP/1.1 \r\n’, ‘:’, ‘\r\n\nHost: ’, ‘\r\n’, ‘:’, ‘\r\n\nHost: ’, ‘\xFF\xBF’, ‘\r\n’</td>
</tr>
<tr>
<td>Token Subsequence</td>
<td>.0008%</td>
<td>0%</td>
<td>GET.* HTTP/1.1 \r\n.<em>\n.</em> \r\n\nHost: .* \r\n.<em>.</em> \r\n\nHost: .* \xFF\xBF.* \r\n</td>
</tr>
<tr>
<td>Bayes</td>
<td>.008%</td>
<td>0%</td>
<td>‘\r\n’: 0.0000, ‘:’: 0.0000, ‘\r\n\nHost: ’: 0.0022, ‘GET’: 0.0035, ‘HTTP/1.1 \r\n’: 0.1108, ‘\xFF\xBF’: 3.1517. Threshold: 1.9934</td>
</tr>
</tbody>
</table>

Table 1. Apache-Knacker signatures. These signatures were successfully generated for innocuous pools containing at least 3 worm samples.
Evaluation

Experiment 1. One worm / No innocuous flows

<table>
<thead>
<tr>
<th>Class</th>
<th>False +</th>
<th>False -</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Longest Substring</td>
<td>.3279%</td>
<td>0%</td>
<td>\x00\x00\xF0A</td>
</tr>
<tr>
<td>Best Substring</td>
<td>.0023%</td>
<td>0%</td>
<td>\xFF\xBF</td>
</tr>
<tr>
<td>Conjunction</td>
<td>0%</td>
<td>0%</td>
<td>‘\xFF\xBF’, ‘\x00\x00\xF0A’</td>
</tr>
<tr>
<td>Token Subsequence</td>
<td>0%</td>
<td>0%</td>
<td>\xFF\xBF.*, \x00\x00\xF0A</td>
</tr>
<tr>
<td>Bayes</td>
<td>.0023%</td>
<td>0%</td>
<td>‘\x00\x00\xF0A’; 1.7574, ‘\xFF\xBF’; 4.3295</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Threshold: 4.2232</td>
</tr>
</tbody>
</table>

Table 2. BIND-TSIG signatures. These signatures were successfully generated for innocuous pools containing at least 3 worm samples.
Evaluation

Experiment 1. One worm / No innocuous flows

- Experiment Summary:
  - Conjunction and token subsequence exhibit significantly fewer false positives when compared to a single substring (longest / best)
  - Token subsequence has lower false positive rate than conjunction due to ordering property
  - Correct signature generation requires suspicious pool size to be greater than 2 (otherwise signatures are too specific)
Evaluation

Experiment 2. One worm / Some innocuous flows

(a) Apache-Knacker exploit
(b) BIND TSIG exploit

Figure 5. False positives due to noise in suspicious pool.
Evaluation

Experiment 2. One worm / Some innocuous flows

• Experiment Summary:
  • Polygraph generates a signature for the 5 worm flows which are correct and no signatures are generated for the innocuous traffic.
  • Bayes performs well until noise reaches 80% of suspicious flow pool (may be adjusted for high noise ratios).
  • With HTTP exploits, when sufficient noise is present signatures are created for clusters of some of the innocuous flows (noise which is significantly different from that in the innocuous pool).
Evaluation

Experiment 3. Multiple Worms / Some Innocuous Flows

Figure 5. False positives due to noise in suspicious pool.
Evaluation

Experiment 3. Multiple Worms / Some Innocuous Flows

- Experiment Summary:
  - Similar to Experiment 2
  - Conjunction and token subsequence signatures were generated for each of the polymorphic worms
  - Bayes generated a single signature for both worms
  - No false positives until noise ratio is high
Attack Analysis

Polygraph-specific attacks

- Coincidental-pattern attack:
  - Attacker's polymorphic engine chooses wildcard bytes from a smaller distribution
  - This causes coincidental substrings to be present which do not occur in every sample of the worm

Figure 6. Number of worm samples required when under ‘coincidental-pattern’ attack.
Attack Analysis

Polygraph-specific attacks

- Red herring attack:
  - Worm initially uses fixed tokens as wildcard bytes
  - After some time, this value is changed causing previously generated signatures to no longer match
  - Bayes is resilient to this type of attack

- Innocuous pool poisoning attack:
  - Attacker creates innocuous flows which the signature for a polymorphic worm will match
  - This will cause the signature to cause a high false positive rate
  - If using clustering, the signature may not be generated to match this worm
  - Otherwise, polygraph may decide that the signature is too general
  - Mitigate by using older innocuous data or distributed innocuous pools
Attack Analysis

Polygraph-specific attacks

• Long-tail attack:
  • Because networks flows are only revealed one packet at a time, it may be possible for a network exploit to occur before the signature can be matched
  • Especially for token subsequence, because of its order requirement, this attack may be mitigated by using only enough of the signature to ensure a low false positive rate
  • Also, buffering the streams until it can be determined whether the flow is a worm or not would fix this problem
Related Work

Signature-based Intrusion Detection Systems (SBSs)

- Pattern-based
  - Honeycomb
  - Autograph
  - Earlybird
  - Polygraph

- Semantic-based
  - TaintCheck
  - *motif-finding* (from computational biology)
Related Work

Signature-based Intrusion Detection Systems (SBSs)

- Honeycomb
  - uses longest common substring (LCS) algorithm
  - generates signatures from *honeypot* traffic (i.e. malicious)

Source: [4]
Related Work

Signature-based Intrusion Detection Systems (SBSs)

- TaintCheck (semantic-based)
  - runs programs to be monitored on an emulator
  - monitors data from the attack payload at the processor-instruction level
  - detects format string attacks
  - detects overwrite attacks
  - works without source code of monitored program (e.g. commercial software)
  - may be used to verify the quality of other automated signatures
Conclusion

- Polygraph generates *quality* signatures *automatically*

- Three classes of signatures
  - Conjunction
  - Token subsequence
  - Bayes

- Which should be used?
  - All have advantages and disadvantages
  - No one algorithm is superior for *every* worm

- Best approach is to use all three and use the *best* signature (fewest false positives and false negatives)
References


Questions?
!!!Beat Florida!!!