Polymorphic Attacks against Sequence-based Software Birthmarks

Hyoungshick Kim\textsuperscript{1}, Wei Ming Khoo\textsuperscript{2}, Pietro Liò\textsuperscript{2}
\textsuperscript{1}University of British Columbia, \textsuperscript{2}University of Cambridge
Software Security and Protection Workshop (SSP’12)
16 June 2012
Background

• A software birthmark is “...a characteristic(s) inherent to a program that uniquely identifies it” (Myles & Collberg, 2004)

• We consider the clone detection problem

Alice
Honest software vendor

Bob
Evil software analyst

$P1 == P2$?
Software birthmark detection

- 2 Phases: Bob first applies birthmarking function `mark()`
- Then applies detection function `detect()`
- Alice wins if $B1 \neq B2$ (!`detect()`) when $P1 == P2$
- Bob wins if $B1 == B2$ (`detect()`) when $P1 == P2$

![Diagram showing the process of marking and detection]
Sequence-based birthmarks

• Well-known birthmarking scheme [Tamada'04, Myles'05, Wang'09]
  – Sequence of API and system calls (or instructions)
  – \textit{Mark}(P) is a sequence of symbols in a finite alphabet \( \Sigma = \{a_1, \ldots, a_k\} \)
  – E.g. \{\texttt{fopen, gettimeofday, fscanf, fclose, ...}\}
Multiple Sequence Alignment (MSA)

• Well-known bioinformatics problem [Higgins'88, Brudno'03, Edgar'04]
• Recently found a use in software birthmarking [Park'08, Wang'09]
• Alignment is a way of arranging two or more sequences to identify regions of similarity/dissimilarity
• Given a set of \( n \) sequences, the goal is to generate an \( n \times n \) distance matrix
Sequence alignment

• Several parameters to optimize
  – Global/Local alignment (ClustalW)
  – Gap opening/extension cost
  – Match/mismatch cost
  – For our purposes, set a threshold distance

- Gaps
  cmp-branch
  fn prologue
  imul
Our contributions

- We show that the intuitive strategies of randomly inserting/deleting symbols are ineffective at defeating sequence alignment-based clone detection, even at high rates.

- Instead we show empirically that non-consecutive insertions and highest frequency deletions are twice as cost-effective.

- We also discuss the costs of such attacks, and propose using non-determinism through concurrent programming as an alternative strategy.
Polymorphic Attacks
A simple attack

- Random Insertion, INS(R)
- Define insertion ratio $x_i \in [0, 2]$
- For a mark($P$) of length $n$, choose $n \cdot x_i$ bogus symbols from $\Sigma$ and insert at random positions of mark($P$)
- Effectiveness?
Test corpus

- FakAV-DO (trojan)
- Skyhoo (trojan)
- Triangle (benign)
- Notepad (benign)
- 7zip (benign)
- WinSCP (benign)

- Pin+VMWare used capture API call traces
- 48 birthmarks, 370 API/system calls

<table>
<thead>
<tr>
<th>Program</th>
<th>n</th>
<th>m</th>
<th>n/m</th>
<th>Entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>triangle</td>
<td>42</td>
<td>21</td>
<td>2.000</td>
<td>4.0736</td>
</tr>
<tr>
<td>notepad</td>
<td>908</td>
<td>81</td>
<td>11.209</td>
<td>4.1500</td>
</tr>
<tr>
<td>fak-do</td>
<td>33</td>
<td>6</td>
<td>5.500</td>
<td>1.8937</td>
</tr>
<tr>
<td>skyhoo</td>
<td>17</td>
<td>4</td>
<td>4.250</td>
<td>1.5197</td>
</tr>
</tbody>
</table>

- $n$ – birthmark length
- $m$ – number of unique symbols
Parameter tuning

- Trained alignment parameters (gap opening, gap extension, mismatch costs), similarity threshold to get birthmark detection rate of 100%

![Graphs showing accuracy, sensitivity, and specificity for different gap opening, gap extension, and mismatch costs.](image-url)
Evaluation

Detection threshold: Similarity score is 0
Can we do better?
Non-consecutive insertion, \textsc{INS}(N)

- Define insertion ratio $x_i \in [0, 2]$

For a \textit{mark}(P) of length $n$, choose $n \times x_i$ bogus symbols from $\Sigma$ and group them into $k$ sequences, $b_1, \ldots, b_k$

- Divide \textit{mark}(P) into sub-sequences $\sigma_1, \ldots, \sigma_k$

Insert $b_i$ at the beginning of $\sigma_i$
Evaluation

INS(N) \sim \text{twice as effective for the same } x_i

How about deletion?
Deletion attacks

- Random Deletion, $\text{DEL}(R)$
- Define deletion ratio $x_d \in [0, 1]$
- For a $\text{mark}(P)$ with $m$ unique symbols, choose $m \times x_d$ symbols and delete them from $\text{mark}(P)$

AB CDEAB CDEAB CDEFABAB CAAB CDAB CDEAB CDEF

$\text{DEL}(R), x_d = 2/6$
Highest frequency deletion, $\text{DEL}(H)$

- Define deletion ratio $x_d \in [0, 1]$

- For a $\text{mark}(P)$ with $m$ unique symbols, choose the $m \times x_d$ highest frequency symbol and delete it from $\text{mark}(P)$

\[ \text{DEL}(H), \quad x_d = \frac{2}{6} \]
How about hybrid attacks – insertion and deletion?
Hybrid attacks

\[
\text{HYB(RR)} = \text{INS}(R) + \text{DEL}(R)
\]

\[
\text{HYB(RN)} = \text{INS}(N) + \text{DEL}(R)
\]

\[
\text{HYB(HR)} = \text{INS}(R) + \text{DEL}(H) \quad \text{(Skyhoo)}
\]

\[
\text{HYB(HN)} = \text{INS}(N) + \text{DEL}(H)
\]
Discussion
Discussion

• How costly are these transformations?
• Depends on
  – What is inserted/deleted
  – Where it is inserted/deleted

Example

```plaintext
push eax;
call IsDebuggerPresent;
pop eax;
```

• Inserting at location 0 is (mostly) free:
  – Packing is a special case of INS(N) with \( k=1 \)
• If a loop occurs \( n \) times, inserting \( i \) in the loop implies inserting \( n \) copies
• Is there an automated way?
Dynamic dependency profiling

• Source-level dependence profiling for estimating potential parallelism (Mak et al. 2010)

• Idea: Use data and control dependencies to identify the critical path of a program

• Tasks not on the critical path can be refactored (within boundaries allowed by dependencies)

• How about exploiting non-determinism?
Concurrency

- Simulate effects of multi-threading on sequence alignment
- Define 100% parallelism as $n$ threads of equal length
- Define 0% parallelism as 1 thread
- However, parallel programming is hard to get correct
- Dummy threads have to factor cost and resiliency
Conclusions & Future work

• Random insertions/deletions were not effective

• HYB(HN) was most cost effective attack strategy

• To look at:
  • Dependency profiling on binaries
  • Static birthmarking schemes
  • Evaluating larger corpus, other code transformations