Quantifying Information Leakage of Probabilistic Programs Using the PRISM Model Checker

K. Salehi, A. A. Noroozi, and S. Amir-Mohammadian

Dept. of Computer Science, Shahrekord University, Shahrekord, Iran
Dept. of Computer Science, University of Tabriz, Tabriz, Iran
Dept. of Computer Science, University of the Pacific, Stockton, CA, USA

The Fifteenth International Conference on Emerging Security Information, Systems and Technologies
SECURWARE 2021
Athens, Greece, November 14-18, 2021
Introduction

Confidentiality
Common mechanisms for confidentiality:

Cryptography               Access control            Firewall
Introduction

Information leakage

secret variables

public variables
Introduction

Information leakage

\[ l := h \mid (1100)_b \]

2 rightmost bits of \( h \) are leaked into \( l \)
Information leakage

while $l_1 < h \mod 2$ do
  $l_1 := l_1 + 1$;
  $l_2 := \text{random}(2)$;
end

1 bit of $h$ is leaked into $l_1$
1. An automated method:

- Modeling programs by Markov chains,
- Computing joint probabilities of the program’s secrets and public outputs,
- Calculating the exact value of information leakage.
Contributions

2. PRISM-Leak

A tool for evaluating secure information flow of concurrent probabilistic programs

- leakage
- prism
- information-leakage
- binary-decision-diagrams
- prism-language
- security
- security-tool
- concurrent-probabilistic-programs

- confidentiality

32 commits
2 branches
2 releases
1 contributor
GPL-3.0

Latest commit ef4571b on Jul 29
- cudd
  - Version 1.1
  - last month
- prism-leak
  - Update conditional probabilities
  - last month
Contributions

3. Case study:

the grades protocol
Contents

1. Introduction
2. The proposed Method
3. Implementation and case study
4. Related work
5. Conclusion

Quantifying Information Leakage of Probabilistic Programs …
The proposed method

Markov Chain \( \mathcal{M} = (S, P, \zeta) \)

\[
\begin{align*}
\text{h = 0} & \quad \langle 0, 0 \rangle \\
\text{s_0} & \quad 1 = \langle 0, 0 \rangle \\
\text{s_1} & \quad 1 = \langle 1, 0 \rangle \\
\text{s_2} & \quad 1 = \langle 1, 0 \rangle \\
\text{s_3} & \quad 1 = \langle 1, 0 \rangle \\
\text{s_4} & \quad 1 = \langle 1, 1 \rangle \\
\end{align*}
\]

\[
\begin{align*}
\text{h = 1} & \quad \langle 0, 0 \rangle \\
\text{s_1} & \quad 1 = \langle 0, 0 \rangle \\
\text{s_2} & \quad 1 = \langle 1, 0 \rangle \\
\text{s_3} & \quad 1 = \langle 1, 0 \rangle \\
\text{s_4} & \quad 1 = \langle 1, 1 \rangle \\
\end{align*}
\]

\[
\begin{align*}
\text{h = 2} & \quad \langle 0, 0 \rangle \\
\text{s_5} & \quad 1 = \langle 0, 0 \rangle \\
\text{s_6} & \quad 1 = \langle 0, 0 \rangle \\
\text{s_7} & \quad 1 = \langle 1, 0 \rangle \\
\text{s_8} & \quad 1 = \langle 1, 0 \rangle \\
\text{s_9} & \quad 1 = \langle 1, 1 \rangle \\
\end{align*}
\]
The proposed method

Markov Chain \( \mathcal{M} = (S, P, \zeta) \)

Quantifying Information Leakage of Probabilistic Programs
The proposed method

Markov Chain \( \mathcal{M} = (S, \mathbf{P}, \zeta) \)

\[
\begin{align*}
\text{h = 0} & \quad s_0 & 1 = \langle 0, 0 \rangle \\
\text{h = 1} & \quad s_1 & 1 = \langle 0, 0 \rangle \\
& \quad s_2 & 1 = \langle 1, 0 \rangle \\
& \quad s_3 & 1 = \langle 1, 0 \rangle \\
& \quad s_4 & 1 = \langle 1, 1 \rangle \\
\text{h = 2} & \quad s_5 & 1 = \langle 0, 0 \rangle \\
\text{h = 3} & \quad s_6 & 1 = \langle 0, 0 \rangle \\
& \quad s_7 & 1 = \langle 1, 0 \rangle \\
& \quad s_8 & 1 = \langle 1, 0 \rangle \\
& \quad s_9 & 1 = \langle 1, 1 \rangle \\
\end{align*}
\]
Preliminaries

Path

\[ \pi = s_1 s_2 s_3 \]
Preliminaries

Occurrence probability of a path

\[
\Pr(\pi = s_1s_2s_3) = 0.25 \times 1 \times 0.5 = 0.125
\]
The proposed method

Information leakage

\[ \mathcal{L}(\mathcal{M}) = \text{initial uncertainty} - \text{remaining uncertainty} \]
The proposed method

Information leakage

\[ \mathcal{L}(\mathcal{M}) = \text{initial uncertainty} - \text{remaining uncertainty} \]

Shannon entropy:

\[ \mathcal{H}(\mathcal{X}) = -\Sigma_{x \in \mathcal{X}} Pr(\mathcal{X} = x) \log_2 Pr(\mathcal{X} = x) \]
The proposed method

Information leakage

\[ \mathcal{L}(\mathcal{M}) = \text{initial uncertainty} - \text{remaining uncertainty} \]

\[ \mathcal{L}(\mathcal{M}) = \mathcal{H}(h) - \mathcal{H}(h | o) \]
Initial uncertainty

\[ \mathcal{H}(h) = - \sum_{\bar{h} \in h} Pr\left(h = \bar{h}\right) \cdot \log_2 Pr\left(h = \bar{h}\right) \]
The proposed method

Remaining uncertainty

\[
\mathcal{H}(h \mid o) = - \sum_{\bar{o} \in o} Pr(o = \bar{o}) \cdot \mathcal{H}(h \mid o = \bar{o})
\]

\[
- \sum_{\bar{h} \in h} Pr(h = \bar{h} \mid o = \bar{o}) \cdot \log_2 Pr(h = \bar{h} \mid o = \bar{o})
\]

\[
\sum_{\bar{h} \in h} Pr(h = \bar{h}, o = \bar{o}) \cdot \frac{Pr(h = \bar{h}, o = \bar{o})}{Pr(o = \bar{o})}
\]
The proposed method

Remaining uncertainty

\[
\mathcal{H}(h \mid o) = - \sum_{\bar{o} \in o} Pr(o = \bar{o}) \cdot \mathcal{H}(h \mid o = \bar{o})
\]

\[
- \sum_{\vec{h} \in h} Pr(h = \bar{h} \mid o = \bar{o}) \cdot \log_2 Pr(h = \bar{h} \mid o = \bar{o})
\]

Quantifying Information Leakage of Probabilistic Programs …
The proposed method

\[ \sum_{h \in h} \Pr(h = \bar{h}, o = \bar{o}) = \sum_{s_0 \in \text{Init}(M), s_n = <\bar{o},\bar{h},...>} \Pr(\pi = s_0 \ldots s_n) \]
The proposed method

Input: finite MC $\mathcal{M}$
Output: a map containing the joint probabilities $Pr(h, o)$

1: Let $ohMap$ be an empty higher-order map function from $\overline{o}$ to $\overline{h}$ to $Pr(h = \overline{h}, o = \overline{o})$;
   // i.e. $ohMap : \overline{o} \mapsto (\overline{h} \mapsto Pr(h = \overline{h}, o = \overline{o}))$
2: Let $\pi$ be an empty list of states for storing a path;
3: for $s_0$ in $Init(\mathcal{M})$ do
4:    EXPLOREPATHS($s_0$, $\pi$, $ohMap$);
5: return $ohMap$;
The proposed method

6: function EXPLORE PATHS(s, π, ohMap)
   // add state s to the current path from the initial state
7:   π.add(s);
   // found a path stored in π
8:   if s is a terminating state then
9:     // assume s = (\overline{\sigma}, \overline{h}, \ldots)
10:    // define hMap as Pr(h, o = \overline{\sigma})
11:   if \overline{\sigma} not in ohMap then
12:     Let hMap be an empty map from
13:     \overline{h} to Pr(h = \overline{h}, o = \overline{\sigma});
14:   else
15:     hMap = ohMap.get(\overline{\sigma});
16:   if \overline{h} not in hMap then
17:     prob = Pr(π);
18:   else
19:     prob = Pr(π) + hMap.get(\overline{h});
20:   end if
21:   hMap.put(\overline{h}, prob); // Update hMap
22:   ohMap.put(\overline{\sigma}, hMap); // Update ohMap
23: else
24:   for s' in Post(s) do
25:     EXPLORE PATHS(s', π, ohMap);
26:     // done exploring from s, so remove it from π
27:     π.pop();
28:   end for
29: end if
30: return ;
The proposed method

Time complexity:

\[ O(2^n) \]
Contents

1. Introduction
2. The proposed Method
3. Implementation and case study
4. Related work
5. Conclusion
Implementation

PRISM-Leak:

Source Code → Label Assignment → Instrumented Source Code → PRISM

Leakage Computation → Leakage Value → Markov Chain
The grades protocol

- $k$ students $s_1, ..., s_k$
- secret grades $g_1, ..., g_k$ where $0 \leq g_i < m$

- Goal: computing sum of the grades, without revealing the secret grades to other students
The grades protocol

- $k$ students $s_1, \ldots, s_k$
- secret grades $g_1, \ldots, g_k$ where $0 \leq g_i < m$

- $n = (m - 1) \times k + 1$
- $r_i \in [0, n]$
- $d_i = g_i + r_i - r_{(i+1)\%k}$
- sum = $(\sum_i d_i) \% n$
### Case study

The grades protocol

<table>
<thead>
<tr>
<th>$m$</th>
<th>$k$</th>
<th>$M_{grades}$</th>
<th>Leakage (bits)</th>
<th>$M_{sum}$</th>
<th>Leakage (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td># states</td>
<td># transitions</td>
<td></td>
<td># states</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>196</td>
<td>228</td>
<td>1.5 (75%)</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3752</td>
<td>4256</td>
<td>1.81 (60.4%)</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>92496</td>
<td>102480</td>
<td>2.03 (50.8%)</td>
<td>256</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1179</td>
<td>1395</td>
<td>2.2 (69.3%)</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>66366</td>
<td>75600</td>
<td>2.53 (53.1%)</td>
<td>216</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>439668</td>
<td>597780</td>
<td>2.75 (43.3%)</td>
<td>1296</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>4048</td>
<td>4816</td>
<td>2.66 (66.4%)</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>455104</td>
<td>519040</td>
<td>2.98 (49.7%)</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3271680</td>
<td>6589440</td>
<td>3.2 (40%)</td>
<td>4096</td>
</tr>
</tbody>
</table>
Contents

1 Introduction
2 The proposed Method
3 Implementation and case study
4 Related work
5 Conclusion
Related work

Chothia et al., 2013

• Tool LeakWatch

• Java programs

• Estimation of the leakage

• Intermediate leakages
Klebanov, 2014

- Symbolic execution and self-composition
- Deterministic programs
- Non-automated method
Biondi et al., 2017

• Tool HyLeak
• Sequential programs
• Estimation of the leakage
• No intermediate leakage
Related work

Salehi et al., 2019

- Evolutionary algorithm
- Channel capacity
- Concurrent probabilistic programs
Conclusion

Proposed approach:

- Formal
- Fully-automatic
- Accurate
Future work

1. Comparing scalability
2. Estimating leakage by statistical methods
3. Analyzing case studies in other application domains
References


Thanks for you attention!