Side Channel Analysis Using a Model Counting Constraint Solver and Symbolic Execution

Tevfik Bultan
Computer Science Department
University of California, Santa Barbara (UCSB)

Joint work with:
Abdulbaki Aydin, Lucas Bang, UCSB
Corina Pasareanu, Quoc-Sang Phan, CMU, NASA
Verification Laboratory (VLab)
University of California, Santa Barbara (UCSB)

- VLab: Research on automated verification, program analysis, formal methods, software engineering, computer security
- Recent research: String analysis, Model counting constraint solvers, Side channel analysis, Data model verification, Web application verification and security
- Always looking for talented and hard working graduate students!
Publications most closely related to this talk

“String Analysis for Side Channels with Segmented Oracles.” Lucas Bang, Abdulbaki Aydin, Quoc-Sang Phan, Corina S. Pasareanu, Tevfik Bultan, FSE’16.

Quantitative Information Flow Problem

Given a program and a secret that the program accesses:

Figure out how much information is leaked about the secret by observing the behavior of the program.
Overview

Program → Symbolic Execution → Path Constraints → Model Counting → Probability Distribution for Observables → Side Channel Analysis → Information Leakage
Overview

Program $\rightarrow$ Symbolic Execution $\rightarrow$ Path Constraints $\rightarrow$ Model Counting $\rightarrow$ Probability Distribution for Observables $\rightarrow$ Side Channel Analysis $\rightarrow$ Information Leakage
A 4-digit PIN Checker

```cpp
bool checkPIN(guess[]) {
    for(i = 0; i < 4; i++)
        if(guess[i] != PIN[i])
            return false;
    return true;
}
```

$P$: PIN, $G$: guess
Symbolic Execution of PIN Checker

```c
bool checkPIN(guess[]) {
    for(i = 0; i < 4; i++)
        if(guess[i] != PIN[i])
            return false;
    return true;
}
```

\( P \): PIN, \( G \): guess
Probabilistic Symbolic Execution

Can we determine the probability of executing a program path?

- Let $PC_i$ denote the path constraint for a program path.
- Let $|PC_i|$ denote the number of possible solutions for $PC_i$.
- Let $|D|$ denote the size of the input domain.
- Assume uniform distribution over the input domain.
- Then the probability of executing that program path is:

$$ p(PC_i) = \frac{|PC_i|}{|D|} $$
Probabilistic Symbolic Execution of PIN Checker

- Assume binary 4 digit PIN, P and G each have 4 bits
- \(|D| = 2^8 = 256\)

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(PC_i)</td>
<td>(P[0] \neq G[0])</td>
<td>(P[0] = G[0])</td>
<td>(P[0] = G[0])</td>
<td>(P[0] = G[0])</td>
<td>(P[0] = G[0])</td>
</tr>
</tbody>
</table>

\(|PC_i|\)

\(p_i\)

- \(p(\text{PC}_i) = |\text{PC}_i| / |D|\)
Probabilistic Symbolic Execution of PIN Checker

- Assume binary 4 digit PIN, P and G each have 4 bits
- \(|D| = 2^8 = 256\

<table>
<thead>
<tr>
<th>(i)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
</table>

| \(|PC_i|\) | 128   |       |       |       |       |
| \(p_i\)   | 1/2   |       |       |       |       |

- \(p(\text{PC}_i) = |\text{PC}_i| / |D|\)
**Probabilistic Symbolic Execution of PIN Checker**

- Assume binary 4 digit PIN, P and G each have 4 bits
- \(|D| = 2^8 = 256\)

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>PC_i</td>
<td>)</td>
<td>128</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>(p_i)</td>
<td>1/2</td>
<td>1/4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \(p(PC_i) = |PC_i| / |D|\)
Probabilistic Symbolic Execution of PIN Checker

- Assume binary 4 digit PIN, P and G each have 4 bits
- $|D| = 2^8 = 256$

<table>
<thead>
<tr>
<th>$i$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PC_i$</td>
<td>$P[0] \neq G[0]$</td>
<td>$P[0] = G[0]$</td>
<td>$P[0] = G[0]$</td>
<td>$P[0] = G[0]$</td>
<td>$P[0] = G[0]$</td>
</tr>
<tr>
<td>$</td>
<td>PC_i</td>
<td>$</td>
<td>128</td>
<td>64</td>
<td>32</td>
</tr>
<tr>
<td>$p_i$</td>
<td>1/2</td>
<td>1/4</td>
<td>1/8</td>
<td>1/16</td>
<td>1/16</td>
</tr>
</tbody>
</table>

- Probability that an adversary can guess a prefix of length $i$ in one guess is given by $p_i$
Overview

Program → Symbolic Execution → Path Constraints → Model Counting → Probability Distribution for Observables → Side Channel Analysis → Information Leakage
Information Leakage

- Note that any PIN checker leaks information about the secret (secret is the pin value P)

- When an adversary tries a guess G there are two scenarios:
  - If G matches P then adversary learns the PIN
  - If G does not match P, then the adversary learns that the PIN value is not G

- This is due to the public output of the PIN checker
  - This is called the main channel

- However, there may be other observations one can make about the PIN checker that reveals more information about P
Information Leakage

- An adversary may observe more than just the public output of a program, such as
  - execution time
  - memory usage
  - file size
  - network package size

- There may be information leakage about the secret from these observable values

- These are called side channels
Entropy: Quantifying Information Leakage

- How can we quantify information leakage?

- Shannon Entropy

\[ H = \sum p_i \log \frac{1}{p_i} = E \left[ \log \frac{1}{p_i} \right] \]

- Intuition:
  - The *expected* amount of *information gain* (i.e., the expected amount of surprise) expressed in terms of *bits*
Entropy: Quantifying Information Leakage

- Entropy example:
  - Seattle weather in December: Always raining
    - \( p_{\text{rain}} = 1, p_{\text{sun}} = 0 \)
    - Entropy: \( H = 0 \)

- San Francisco weather in December: Coin flip
  - \( p_{\text{rain}} = \frac{1}{2}, p_{\text{sun}} = \frac{1}{2} \)
  - Entropy: \( H = 1 \)

- Santa Barbara weather in December: Almost always beautiful:
  - \( p_{\text{rain}} = \frac{1}{10}, p_{\text{sun}} = \frac{9}{10} \)
  - Entropy: \( H = 0.496 \)
Information Leakage via Side Channels

- Side channels produce a set of observables that partition the secret:  \[ O = \{o_1, o_2, \ldots o_m\} \]

- By computing the probability of observable values we can compute the entropy:
  \[ H(P) = - \sum_{i=1}^{m} p(o_i) \log_2(p(o_i)) \]

- We can compute the probability of observable values using model counting:
  
  the probability of observing \( o_i \) is:
  \[ p(o_i) = \frac{\sum \#(PC_j(h, l))}{\#D} \text{ where } cost(\pi_j) = o_i \]
Symbolic Execution of PIN Checker

```c
bool checkPIN(guess[])
for(i = 0; i < 4; i++)
    if (guess[i] != PIN[i])
        return false
return true
```

\( P \): PIN, \( G \): guess
\( o_i \) = lines of code

\( o_0 = 3 \)
\( o_1 = 5 \)
\( o_2 = 7 \)
\( o_3 = 9 \)
\( o_4 = 10 \)
Probabilistic Symbolic Execution of PIN Checker

- Assume binary 4 digit PIN, P and G each have 4 bits
- $|D| = 2^8 = 256$

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PC_i$</td>
<td>$P[0] \neq G[0]$</td>
<td>$P[0] = G[0]$</td>
<td>$P[0] = G[0]$</td>
<td>$P[0] = G[0]$</td>
<td>$P[0] = G[0]$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>return</th>
<th>false</th>
<th>false</th>
<th>false</th>
<th>false</th>
<th>true</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>PC_i</td>
<td>$</td>
<td>128</td>
<td>64</td>
<td>32</td>
</tr>
<tr>
<td>$p_i$</td>
<td>$1/2$</td>
<td>$1/4$</td>
<td>$1/8$</td>
<td>$1/16$</td>
<td>$1/16$</td>
</tr>
<tr>
<td>$o_i$</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>
### Information Leakage

<table>
<thead>
<tr>
<th>$i$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>return</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>$</td>
<td>PC_i</td>
<td>$</td>
<td>128</td>
<td>64</td>
<td>32</td>
</tr>
<tr>
<td>$p_i$</td>
<td>$1/2$</td>
<td>$1/4$</td>
<td>$1/8$</td>
<td>$1/16$</td>
<td>$1/16$</td>
</tr>
<tr>
<td>$o_i$</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

\[
H = \sum p_i \log \frac{1}{p_i} = 1.8750
\]

- $H$: The expected amount of information gain by the adversary

---

Bang et al., String Analysis for Side Channels with Segmented Oracles (FSE’16)
A secure PIN checker

```java
public verifyPassword (guess[]) {
    matched = true
    for (int i = 0; i < 4; i++)
        if (guess[i] != PIN[i])
            matched = false
    else
        matched = matched
    return matched
}
```

- Only two observables (just the main channel, no side channel): $o_0$: does not match, $o_1$: full match
- $p(o_0) = 15/16$, $p(o_1) = 1/16$
- $H_{secure} = 0.33729$

Bang et al., String Analysis for Side Channels with Segmented Oracles (FSE’16)
Secure vs. insecure PIN checker

- Given a PIN of length $L$ where each PIN digit has $K$ values
- Secure PIN checker
  - $K^L$ guesses in the worst case
  - Example: 16 digit password where each digit is ASCII
  - $128^{16}$ tries in the worst case, which would take a lot of years
- Insecure PIN checker
  - A **prefix attack** that determines each digit one by one starting with the leftmost digit
  - Example: 16 digit password where each digit is ASCII
  - $128 \times 16$ tries in the worst case, which would not take too much time

---

Bang et al., String Analysis for Side Channels with Segmented Oracles (FSE’16)
Secure vs. insecure PIN checker

![Graph showing entropy vs. password length for two functions, $F_1$ and $F_2$.]

Bang et al., String Analysis for Side Channels with Segmented Oracles (FSE’16)
Not just a toy example

Vulnerabilities that are similar to the simple PIN example happen in real software systems

Timing Side Channels

- HMAC keys: Google Keyczar Library, Xbox 360
- Authorization Frameworks: OAuth, OpenID
- Java’s Array.equals, String.equals
- C’s memcmp

Network Packet Size Side Channel

- Compression Ratio Infoleak Made Easy (CRIME)

Bang et al., String Analysis for Side Channels with Segmented Oracles (FSE’16)
Overview

Program → Symbolic Execution → Path Constraints → Model Counting → Probability Distribution for Observables → Side Channel Analysis → Information Leakage
Model Counting String Constraint Solver

**INPUT**

- string constraint: $C$

**OUTPUT**

- counting function: $f_C$
- length bound: $k$

- # of strings with length $\leq k$ for which $C$ evaluates to true

---

Aydin et al., Automata-based Model Counting for String Constraints. (CAV'15)
Automata Based Counter (ABC)
A Model Counting String Constraint Solver

INPUT
string constraint: $C$

OUTPUT
counting function: $f_c$
length bound: $k$

$\#$ of strings with length $\leq k$
for which $C$ evaluates to true

Aydin et al., Automata-based Model Counting for String Constraints. (CAV'15)
String Constraint Language

\[
C \quad \rightarrow \quad \text{bterm}
\]

\[
\text{bterm} \quad \rightarrow \quad v \mid \text{true} \mid \text{false} \\
\quad \mid \neg \text{bterm} \mid \text{bterm} \wedge \text{bterm} \mid \text{bterm} \vee \text{bterm} \mid (\text{bterm}) \\
\quad \mid \text{stern} = \text{stern} \\
\quad \mid \text{match} (\text{stern}, \text{stern}) \\
\quad \mid \text{contains} (\text{stern}, \text{stern}) \\
\quad \mid \text{begins} (\text{stern}, \text{stern}) \\
\quad \mid \text{ends} (\text{stern}, \text{stern}) \\
\quad \mid \text{iterm} = \text{iterm} \mid \text{iterm} \lt \text{iterm} \mid \text{iterm} \gt \text{iterm}
\]

\[
\text{iterm} \quad \rightarrow \quad v \mid n \\
\quad \mid \text{iterm} + \text{iterm} \mid \text{iterm} - \text{iterm} \mid \text{iterm} \times n \mid (\text{iterm}) \\
\quad \mid \text{length} (\text{stern}) \mid \text{toint} (\text{stern}) \\
\quad \mid \text{indexOf} (\text{stern}, \text{stern}) \\
\quad \mid \text{lastIndexOf} (\text{stern}, \text{stern})
\]

\[
\text{stern} \quad \rightarrow \quad v \mid \varepsilon \mid s \\
\quad \mid \text{stern.stern} \mid \text{stern|stern} \mid \text{stern*} \mid (\text{stern}) \\
\quad \mid \text{charAt} (\text{stern}, \text{iterm}) \mid \text{toString} (\text{iterm}) \\
\quad \mid \text{toupper} (\text{stern}) \mid \text{tolower} (\text{stern}) \\
\quad \mid \text{substring} (\text{stern}, \text{iterm}, \text{iterm}) \\
\quad \mid \text{replaceFirst} (\text{stern}, \text{stern}, \text{stern}) \\
\quad \mid \text{replaceLast} (\text{stern}, \text{stern}, \text{stern}) \\
\quad \mid \text{replaceAll} (\text{stern}, \text{stern}, \text{stern})
\]
## Example String Expressions

<table>
<thead>
<tr>
<th>String Expression</th>
<th>Constraint Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.length()</td>
<td>length(s)</td>
</tr>
<tr>
<td>s.isEmpty()</td>
<td>length(s) == 0</td>
</tr>
<tr>
<td>s.startsWith(t,n)</td>
<td>0 ≤ n ∧ n ≤</td>
</tr>
<tr>
<td>s.indexOf(t,n)</td>
<td>indexof(substring(s,n,</td>
</tr>
<tr>
<td>s.replaceAll(p,r)</td>
<td>replaceall(s,p,r)</td>
</tr>
<tr>
<td>strrpos(s, t)</td>
<td>lastindexof(s,t)</td>
</tr>
<tr>
<td>substr_replace(s, t, i, j)</td>
<td>substring(s,0,i).t.substring(s,j,</td>
</tr>
<tr>
<td>strip_tags(s)</td>
<td>replaceall(s,(&quot;&lt;a&gt;&quot;</td>
</tr>
<tr>
<td>mysql_real_escape_string(s)</td>
<td>...replaceall(s ,replaceall(s,&quot;&quot;&quot;,&quot;\&quot;&quot;&quot;) ,&quot;,&quot;&quot;, &quot;&quot;&quot;)...</td>
</tr>
</tbody>
</table>
Model Counting String Constraint Solver

**INPUT**
- string constraint: \( C \)

**Automata-Based model Counting string constraint solver (ABC)**

**OUTPUT**
- counting function: \( f_c \)
- length bound: \( k \)
- \# of strings with length \( \leq k \) for which \( C \) evaluates to true

---

Aydin et al., Automata-based Model Counting for String Constraints. (CAV'15)
String Automata Construction

\[ C \equiv \neg (x \in (01)^*) \land \text{LEN}(x) = 2 \]
String Automata Construction

\[ C \equiv \neg (x \in (01)^*) \land \text{LEN}(x) = 2 \]
String Automata Construction

\[ C \equiv \neg(x \in (01)^*) \land LEN(x) = 2 \]
String Automata Construction

\[ C \equiv \neg (x \in (01)^*) \land LEN(x) = 2 \]
String Automata Construction

\[ C \equiv \neg (x \in (01)^*) \land \text{LEN}(x) = 2 \]
String Automata Construction

\[ C \equiv \neg (x \in (01)^*) \land LEN(x) = 2 \]
String Automata Construction

\[ C \equiv \neg (x \in (01)^*) \land LEN(x) = 2 \]
String Automata Construction

\[ C \equiv \neg (x \in (01)^*) \land \text{LEN}(x) = 2 \]
\[ C \equiv \neg (x \in (01)^*) \land \text{LEN}(x) = 2 \]
String Automata Construction

\[ C \equiv \neg(x \in (01)^*) \land LEN(x) = 2 \]
String Automata Construction

\[ C \equiv \neg(x \in (01)^*) \land LEN(x) = 2 \]
$C \equiv \neg(x \in (01)^*) \land LEN(x) = 2$
String Automata Construction

\[ C \equiv \neg (x \in (01)^*) \land LEN(x) = 2 \]
String Automata Construction

\[ C \equiv \neg (x \in (01)^*) \land LEN(x) = 2 \]
String Automata Construction

\[ C \equiv \neg(x \in (01)^*) \land LEN(x) = 2 \]
String Automata Construction

\[ C \equiv \neg (x \in (01)^*) \land LEN(x) = 2 \]
String Automata Construction

\[ C \equiv \neg (x \in (01)^*) \land LEN(x) = 2 \]
String Automata Construction

\( C \equiv \neg (x \in (01)^*) \land LEN(x) = 2 \)
String Automata Construction

\[ C \equiv \neg (x \in (01)^*) \land LEN(x) = 2 \]
String Automata Construction

\[ C \equiv \neg (x \in (01)^*) \land \text{LEN}(x) = 2 \]
String Automata Construction

\[ C \equiv \neg (x \in (01)^*) \land LEN(x) = 2 \]
String Automata Construction

$C \equiv \neg(x \in (01)^*) \land \text{LEN}(x) = 2$

$00, 10, 11$
Integer Constraints

\[ C \rightarrow bterm \]

\[ bterm \rightarrow v \mid \text{true} \mid \text{false} \]
\[ \quad \rightarrow \neg bterm \mid bterm \land bterm \mid bterm \lor bterm \mid (bterm) \]
\[ \quad \rightarrow \text{stem} = \text{stem} \]
\[ \quad \rightarrow \text{match}(\text{stem}, \text{stem}) \]
\[ \quad \rightarrow \text{contains}(\text{stem}, \text{stem}) \]
\[ \quad \rightarrow \text{begins}(\text{stem}, \text{stem}) \]
\[ \quad \rightarrow \text{ends}(\text{stem}, \text{stem}) \]
\[ \quad \rightarrow \text{iterm} = \text{iterm} \mid \text{iterm} < \text{iterm} \mid \text{iterm} > \text{iterm} \]

\[ iterm \rightarrow v \mid n \]
\[ \quad \rightarrow \text{iterm} + \text{iterm} \mid \text{iterm} - \text{iterm} \mid \text{iterm} \times n \mid (iterm) \]
\[ \quad \rightarrow \text{length}(\text{stem}) \mid \text{toint}(\text{stem}) \]
\[ \quad \rightarrow \text{indexof}(\text{stem}, \text{stem}) \]
\[ \quad \rightarrow \text{lastindexof}(\text{stem}, \text{stem}) \]

\[ stem \rightarrow v \mid \epsilon \mid s \]
\[ \quad \rightarrow \text{stem}.\text{stem} \mid \text{stem}\text{stem} \mid \text{stem}^* \mid (stem) \]
\[ \quad \rightarrow \text{charat}(\text{stem}, \text{iterm}) \mid \text{tostream}(\text{iterm}) \]
\[ \quad \rightarrow \text{toupper}(\text{stem}) \mid \text{tolower}(\text{stem}) \]
\[ \quad \rightarrow \text{substring}(\text{stem}, \text{iterm}, \text{iterm}) \]
\[ \quad \rightarrow \text{replacefirst}(\text{stem}, \text{stem}, \text{stem}) \]
\[ \quad \rightarrow \text{replaceall}(\text{stem}, \text{stem}, \text{stem}) \]
Integer Automata Construction

\[ C \equiv x = -1 \land x + y = 1 \]
Integer Automata Construction

\[ C \equiv x = -1 \land x + y = 1 \]
\[ C_1 \equiv x + 0 \ast y + 1 = 0 \Rightarrow [1 0 1] \]
\[ C_2 \equiv x + y - 1 = 0 \Rightarrow [1 1 - 1] \]
Integer Automata Construction

\[ C \equiv x = -1 \land x + y = 1 \]

\[ C_1 \equiv x + 0 \ast y + 1 = 0 \Rightarrow [1 \ 0 \ 1] \]

\[ C_2 \equiv x + y - 1 = 0 \Rightarrow [1 \ 1 \ -1] \]

\[ C_1 \land C_2 \]

Using automata construction techniques described in:
Conjunction and disjunction is handled by automata product, negation is handled by automata complement
Model Counting String Constraints Solver

**INPUT**

- string constraint: \( C \)

**OUTPUT**

- counting function: \( f_c \)
- length bound: \( k \)

\# of strings with length \( \leq k \) for which \( C \) evaluates to true

---

Aydin et al., Automata-based Model Counting for String Constraints. (CAV'15)
Can you solve it Will Hunting?

Given the graph

\[ \begin{array}{c}
1 \\
2 \\
3 \\
\end{array} \]

Find:
1) the adjacency matrix \( A \)
2) the matrix giving the number of 3 step walks
3) the generating function for walks from point \( i \rightarrow j \)
4) the generating function for walks from points \( 1 \rightarrow 3 \)
Automata-based Model Counting

- Converting constraints to automata reduces the model counting problem to path counting problem in graphs.

We will generate a function $f(k)$

- Given length bound $k$, it will count the number of paths with length $k$.

  - $f(0) = 0, \emptyset$
  - $f(1) = 2, \{0,1\}$
  - $f(2) = 3, \{00,10,11\}$

\[ C \equiv \neg (x \in (01)^*) \]
Path Counting via Matrix Exponentiation

\[ C = \neg (x \in (01)^* ) \]

\[
T = \begin{bmatrix}
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 \\
0 & 0 & 2 & 1 \\
0 & 0 & 0 & 0 \\
\end{bmatrix},
T^2 = \begin{bmatrix}
1 & 0 & 1 & 2 \\
0 & 1 & 3 & 1 \\
0 & 0 & 4 & 2 \\
0 & 0 & 0 & 0 \\
\end{bmatrix},
T^3 = \begin{bmatrix}
0 & 1 & 7 & 3 \\
1 & 0 & 7 & 4 \\
0 & 0 & 8 & 4 \\
0 & 0 & 0 & 0 \\
\end{bmatrix},
T^4 = \begin{bmatrix}
0 & 1 & 15 & 8 \\
0 & 1 & 15 & 7 \\
1 & 0 & 15 & 7 \\
0 & 0 & 16 & 8 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[ f(0) = 0 \]
\[ f(1) = 2 \]
\[ f(2) = 3 \]
\[ f(3) = 8 \]
Path Counting via Recurrence Relation

\[ f(n, k) = \sum_{(m,n) \in E} f(m, k - 1) \]

\[ f(0, 0) = 1 \]
\[ f(1, 0) = 0 \]
\[ f(2, 0) = 0 \]
\[ \ldots \]
\[ f(i, 0) = 0 \]
Path Counting via Recurrence Relation

\[ f(4, k) = f(2, k - 1) + f(3, k - 1) \]
\[ f(3, k) = f(1, k - 1) + f(2, k - 1) + f(3, k - 1) \]
\[ f(2, k) = f(1, k - 1) \]
\[ f(1, k) = f(2, k - 1) \]
\[ f(1, 0) = 1, f(2, 0) = 0, f(3, 0) = 0, f(4, 0) = 0 \]
Path Counting via Recurrence Relation

- We can solve system of recurrence relations for final node

\[ f(0) = 0, \ f(1) = 2, \ f(2) = 3 \]
\[ f(k) = 2f(k-1) + f(k-2) - 2f(k-3) \]
We can compute a generating function, \( g(z) \), for a DFA from the associated matrix \( T \):

\[
T = \begin{bmatrix}
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 \\
0 & 0 & 2 & 1 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
g(z) = (-1)^n \frac{\det(I - zT: n + 1,1)}{z \times \det(I - zT)} = \frac{2z - z^2}{1 - 2z - z^2 + 2z^3}
\]
Counting Paths via Generating Functions

\[ g(z) = \frac{2z - z^2}{1 - 2z - z^2 + 2z^3} \]

Each \( f(i) \) can be computed by Taylor expansion of \( g(z) \)

\[ g(z) = \frac{g(0)}{0!} z^0 + \frac{g(1)(0)}{1!} z^1 + \frac{g(2)(0)}{2!} z^2 + \ldots + \frac{g(n)(0)}{n!} z^n + \ldots \]

\[ g(z) = 0z^0 + 2z^1 + 3z^2 + 8z^3 + 15z^4 + \ldots \]

\[ g(z) = f(0)z^0 + f(1)z^1 + f(2)z^2 + f(3)z^3 + f(4)z^4 + \ldots \]
Good job Will Hunting!

This is correct. Who did this?
Applicable to Both Automata

- Multi-track Binary Integer Automaton:

- String Automaton:
Model Counting String Constraints Solver

**INPUT**

string constraint: $C$

**OUTPUT**

Automata-Based model Counting string constraint solver (ABC)

Counting function: $f_c$

length bound: $k$

# of strings with length $\leq k$ for which $C$ evaluates to true

Aydin et al., Automata-based Model Counting for String Constraints. (CAV’15)
Overview

Program \[\rightarrow\] Symbolic Execution

\[\rightarrow\] Path Constraints

Model Counting

\[\rightarrow\] Probability Distribution for Observables

Side Channel Analysis

\[\rightarrow\] Information Leakage
A case study

• A web service with a database that contains restricted & unrestricted employee IDs
• Supports SEARCH & INSERT queries

• Question: Is there a side channel in time that a third party can determine the value of a single restricted ID in the database
Code Inspection

- Using code inspection we identified that the SEARCH and INSERT operations are implemented in:

```java
class UDPServerHandler

method channelRead0

switch case 1: INSERT

switch case 8: SEARCH
```
public class Driver {
    public static void main(String[] args) {
        BTree tree = new BTree(10);
        CheckRestrictedID checker = new CheckRestrictedID();
        // create two concrete unrestricted ids
        int id1 = 64, id2 = 85;
        tree.add(id1, null, false);
        tree.add(id2, null, false);
        // create one symbolic restricted id
        int h = Debug.makeSymbolicInteger("h");
        Debug.assume(h != id1 && h != id2);
        tree.add(h, null, false);
        checker.add(h);
        UDPServerHandler handler = new UDPServerHandler(tree, checker);
        int key = Debug.makeSymbolicInteger("key");
        handler.channelRead0(8, key);  // send a search query with
                                         // with search range 50 to 100
    }
}
SPF Output

>>>>> There are 5 path conditions and 5 observables

<table>
<thead>
<tr>
<th>Cost</th>
<th>Count</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>9059</td>
<td>15</td>
<td>0.014677</td>
</tr>
<tr>
<td>8713</td>
<td>20</td>
<td>0.019569</td>
</tr>
<tr>
<td>7916</td>
<td>923</td>
<td>0.903131</td>
</tr>
<tr>
<td>8701</td>
<td>14</td>
<td>0.013699</td>
</tr>
<tr>
<td>7951</td>
<td>50</td>
<td>0.048924</td>
</tr>
</tbody>
</table>

Domain Size: 1022
Single Run Leakage: 0.6309758112933285
Observation & Proposed Attack

- SEARCH operation:

  *takes longer when the secret is within the search range*  
  (9059, 8713, 8701 byte code instructions)

  as opposed to the case when the secret is out of the search range (7916, 7951 byte code instructions)

- Proposed attack:

  Measure the time it takes for the search operation to figure out if there is a secret within the search range.
Attack

- Binary search on the ranges of the IDs
- Send two search queries at a time and compare their execution time.
- Refine the search range based on the result.

```c
min = 0; max = MAX_ID // assume MAX_ID is a power of 2
while (min < max)
{
    half = (max-min-1)/2;
    if (time(search(min.. min+half-1)) > time(search(min+half .. max)))
        max = min+half-1;
    else
        min = min+half;
}
```
Attack Output

Running [0, 40000000] at 0.
Comparing 467821 vs 612252...
Comparing 400377 vs 333665...
Comparing 200603 vs 237025...
Running [25000000, 30000000] at 6.
Comparing 163564 vs 115072...
Running [25000000, 27500000] at 8.
Comparing 95736 vs 37388...
Running [25000000, 26250000] at 10.
Comparing 85305 vs 30118...
Running [25000000, 25625000] at 12.
Comparing 22765 vs 72958...
Comparing 2147483647 vs 19353...
Running [25312500, 25468750] at 16.
Comparing 517 vs 2147483647...
Running [25390625, 25468750] at 18.
Comparing 317 vs 2147483647...
Running [25429687, 25468750] at 20.
Comparing 2147483647 vs 302...
Running [25429687, 25449218] at 22.
Comparing 2147483647 vs 287...
Comparing 336 vs 2147483647...

Comparing 300 vs 2147483647...
Running [25437010, 25439452] at 28.
Comparing 2147483647 vs 265...
Comparing 2147483647 vs 328...
Running [25437010, 25437620] at 32.
Comparing 280 vs 2147483647...
Running [25437315, 25437620] at 34.
Comparing 293 vs 2147483647...
Running [25437467, 25437620] at 36.
Comparing 2147483647 vs 281...
Running [25437467, 25437543] at 38.
Comparing 2147483647 vs 613...
Running [25437467, 25437505] at 40.
Comparing 2147483647 vs 258...
Running [25437467, 25437486] at 42.
Comparing 2147483647 vs 291...
Running [25437467, 25437476] at 44.
Comparing 362 vs 2147483647...
Running [25437471, 25437476] at 46.
Comparing 311 vs 2147483647...
Running [25437473, 25437476] at 48.
Comparing 2147483647 vs 2147483647...
Checking oracle for: 25437474...true
Checking oracle for: 25437475...false
Multi-Run Analysis

- The side channel analysis I discussed so far is for analyzing a single execution of a program

- Can we do model multi-run analysis?

- Adversary runs the program on multiple inputs one after another

- Can we determine the amount of information leakage in such a scenario?
Multi-Run Analysis

• For multi-run analysis we need an adversary model
  • Adversary behavior influences the analysis

• It would make sense to calculate the leakage for the best adversary

• For a class of side channels called “segmented oracles” we can use symbolic execution and entropy calculation from a single run to compute the change in the entropy for multiple runs

• This can be used to automatically compute how many tries it will take to reveal the secret.
Results for Password Check

Results for 4 segments with 4 values (8 bits of information)
Results for CRIME

Results for 3 segments with 4 values (6 bits of information)
Noisy Observations

- Entropy computations we have shown so far do not take observation noise into account

- One approach we are investigating to handle noise:
  - Assume a noise distribution (for example normal distribution)
  - Run fuzzing to observe parameters of the distribution (mean and standard deviation)
  - Update entropy calculations using the noise model
Noisy Observation Simulation

Simulated Data, sigma = 1

Corrected Probability Model, Conditional Entropy = 1.75
Noisy Observation Simulation

Simulated Data, sigma = 4

Corrected Probability Model, Conditional Entropy = 1.2801
Conclusions

• By combining symbolic execution with model counting constraint solvers we can quantify information leakage in programs

• We can detect non-trivial side channel vulnerabilities using this approach
Current & Future Work

- More efficient model counting
- More expressive model counting
- Handling noise in observations
- Attack synthesis
Related work: Quantitative Information Flow

Related work: Model Counting

- “Effective lattice point counting in rational convex polytopes.” Jesús A. De Loerab, Raymond Hemmeckeb, Jeremiah Tauzera, Ruriko Yoshidab.
- “From Weighted to Unweighted Model Counting.” Supratik Chakraborty, Dror Fried, Kuldeep S. Meel, Moshe Y. Vardi.
- “Algorithmic Improvements in Approximate Counting for Probabilistic Inference.” From Linear to Logarithmic SAT Calls Supratik Chakraborty, Kuldeep S. Meel, Moshe Y. Vardi.