All You Ever Wanted to Know About Dynamic Taint Analysis & Forward Symbolic Execution (but might have been afraid to ask)

(Yes, we were trying to overflow the title length field on the submission server)

Edward J. Schwartz, Thanassis Avgerinos, David Brumley
A Few Things You Need to Know About Dynamic Taint Analysis & Forward Symbolic Execution (but might have been afraid to ask)

Edward J. Schwartz, Thanassis Avgerinos, David Brumley
The Root of All Evil

Humans write programs

This Talk:
Computers Analyzing Programs Dynamically at Runtime
Two Essential Runtime Analyses

**Dynamic Taint Analysis:**
What values are derived from user input?

- Detect Exploits
- Detect packing in malware
  [Bayer2009, Yin2007]

**Forward Symbolic Execution:**
What input will make execution reach *this* line of code?

- Automated Test Case Generation
- Input Filter Generation
  [Costa2007, Brumley2008]
Our Contributions

1: Turn English descriptions into an algorithm
   – Operational Semantics

2: Algorithm highlights caveats, issues, and unsolved problems that are deceptively hard

Computers Analyzing Programs Dynamically at Runtime

Dynamic Taint Analysis: Is this value affected by user input?

Forward Symbolic Execution: What input will make execution reach this line of code?
Our Contributions (cont’d)

3: Systematize recurring themes in a wealth of previous work
Dynamic Taint Analysis: What values are derived from user input?

1. How it works – example

2. Desired properties

3. Example issue. Paper has many more.
\[ x = \text{get\_input}(\text{src}) \]
\[ y = x + 42 \]
\[ \text{goto } y \]

Input is tainted

\[
\begin{array}{c|c}
\text{Var} & \text{Val} \\
\hline
x & 7 \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{Var} & \text{Tainted?} \\
\hline
x & T \\
\end{array}
\]

Input: \[ t = \text{IsUntrusted(src)} \]
\[ \text{get\_input(src)} \rightarrow t \]
\( x = \text{get\_input}() \)

\( y = x + 42 \)

\(...\)

\( \text{goto } y \)

Data derived from user input is tainted

Taint Propagation

\[
\begin{align*}
\text{BinOp} & \quad t_1 = \tau[x_1] , t_2 = \tau[x_2] \\
x_1 + x_2 & \downarrow t_1 \lor t_2
\end{align*}
\]
Policy Violation Detected

\[ x = \text{get\_input}( \) \]

\[ y = x + 42 \]

\[ \ldots \]

\[ \text{goto } y \]

Taint Checking

\[ P_{\text{goto}}(t_a) = \neg t_a \]

(Must be true to execute)
\[ x = \text{get\_input}() \]
\[ y = \ldots \]
\[ \ldots \]
\[ \text{goto}\ y \]

Jumping to overwritten return address

Different Use: Exploit Detection

Real Use: Program Control

\[
\ldots
\text{strcpy(buffer,argv[1])} ;
\ldots
\text{return} ;
\]
# Memory Load

## Variables

<table>
<thead>
<tr>
<th>Δ</th>
<th>Var</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(x)</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>τ</th>
<th>Var</th>
<th>Tainted?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(x)</td>
<td>T</td>
</tr>
</tbody>
</table>

## Memory

<table>
<thead>
<tr>
<th>(\mu)</th>
<th>Addr</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td>42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\tau_\mu)</th>
<th>Addr</th>
<th>Tainted?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td>F</td>
</tr>
</tbody>
</table>
Problem: Memory Addresses

\[ x = \text{get\_input}( ) \]
\[ y = \text{load}(x) \]

[Diagram with a devil character and arrows pointing to the variables]

... goto y

All values derived from user input are tainted??

<table>
<thead>
<tr>
<th>( \Delta )</th>
<th>Var</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \mu )</th>
<th>Addr</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td>42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \tau_{\mu} )</th>
<th>Addr</th>
<th>Tainted?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td>F</td>
</tr>
</tbody>
</table>
Policy 1: Taint depends only on the memory cell

\[ \mu \]

\[
\begin{array}{c}
\text{x} = \text{get\_input} \\
\text{y} = \text{load(\text{x})} \\
\text{goto} \ \text{y}
\end{array}
\]

Taint Propagation

\[
\begin{align*}
\text{Load} & \quad v = \Delta[x], \ t = \tau_\mu[v] \\
\text{load(x)} & \downarrow t
\end{align*}
\]

Undertainting

Failing to identify tainted values - e.g., missing exploits

\[
\begin{array}{|c|c|}
\hline
\text{Addr} & \text{Val} \\
\hline
7 & 7 \\
7 & 42 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{Addr} & \text{Tainted?} \\
\hline
7 & \text{F} \\
\hline
\end{array}
\]
Policy 2: If either the address or the memory cell is tainted, then the value is tainted.

\[ x = \text{get_input} \]
\[ y = \text{load}(\text{jmp_table} + x \mod 2) \]
\[ \text{...} \]
\[ \text{goto } y \]

Overtainting

Unaffected values are tainted
- e.g., exploits on safe inputs

Taint Propagation

\[ v = \Delta[x], \ t = \tau_{\mu}[v], \ t_a = \tau[x] \]
\[ \text{load}(x) \downarrow t \ v \ t_a \]
Research Challenge
State-of-the-Art is not perfect for all programs

Undertainting: Policy may miss taint

Overtainting: Policy may wrongly detect taint
Forward Symbolic Execution:
What input will make execution reach \textit{this} line of code?

- How it works – example
- Inherent problems of symbolic execution
- Proposed solutions
The Challenge

packet_len(int header, char *packet)
char buf[2048] = "...";
if (header < 0)
    return 0;
if (header == 0x12345678)
    strcpy(buf, packet);
return strlen(buf);

Forward Symbolic Execution:
What input will make execution reach this line of code?
A Simple Example

What input will make execution reach this line of code?

If (header < 0)

If (header == 0x12345678)

return 0;

strcpy(buf, packet);

If (header ≥ 0 ∨ header != 0x12345678)

return strlen(buf);

If (header ≥ 0 ∧ header != 0x12345678)

strcpy(buf, packet);

If (header ≥ 0 ∧ header == 0x12345678)

return 0;

If (header < 0)

return 0;
One Problem: Exponential Blowup Due to Branches

Exponential Number of Interpreters/formulas in # of branches
Path Selection Heuristics

Symbolic Execution Tree

However, these are heuristics. In the worst case all create an exponential number of formulas in the tree height.

- Depth-First Search (bounded), Random Search [Cadar2008]
- Concolic Testing [Sen2005, Godefroid2008]
Symbolic Execution is *not* Easy

- Exponential number of interpreters/formulas
- Exponentially-sized formulas
- Solving a formula is NP-Complete!
Other Important Issues

Symbolic Memory

\( \Pi = (s + s + s + s + s + s + s + s) = 42 \)

FORWARD SYMBOLIC EXECUTION

"Like normal execution, where inputs are substituted by symbolic variables"

King et al., 1976

Sanitization

--

Formalization

More complex policies

Symbolic Jumps

If symbolic then goto a
Else goto b

goto symbolic
Conclusion

• Dynamic taint analysis and forward symbolic execution used extensively in literature
  – Formal algorithm and what is done for each possible step of execution often not emphasized

• We provided a formal definition and summarized
  – Critical issues
  – State-of-the-art solutions
  – Common tradeoffs
Thank You!
thanassis@cmu.edu

Questions?