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 packing in malware: [Bayer2009, Yin2007]

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


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

Our Contributions (cont’d)

3: Systematize recurring themes in a wealth of previous work

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?
1. How it works – example
2. Desired properties
3. Example issue. Paper has many more.

```c
x = get_input()
y = x + 42
... goto y

Input is tainted
```

**Taint Introduction**

Input `t = IsUntrusted(src)`

**Taint Propagation**

Input is tainted

```c
x = get_input()
y = x + 42
... goto y
```

Policy Violation Detected

```c
\text{goto (ta) } = \neg t_a \quad \text{(Must be true to execute)}
```

**Taint Checking**

Real Use: Exploit Detection

```c
\text{strcpy(buffer,argv[1])}
\text{return ;}
```

**Memory Load**

```c
\text{... strcpy(buffer,argv[1]) ; ... return ;}
```
Problem: Memory Addresses

All values derived from user input are tainted??

Policy 1: Taint depends only on the memory cell

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

Research Challenge
State-of-the-Art is not perfect for all programs

Policy 2: If either the address or the memory cell is tainted, then the value is tainted

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

The Challenge

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

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 \( x \leq 0 \)
\[
\text{return } -x
\]

If \( x = 0x12345678 \)
\[
\text{return } -x
\]

If \( x = 0x12345678 \)
\[
\text{return } x
\]

One Problem:
Exponential Blowup Due to Branches

Symbolic Execution is not Easy

Path Selection Heuristics

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 [Ben2005, Godefroid2008]

Other Important Issues

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!
thanasis@cmu.edu

Questions?