CS 261: Systems Security

Taint Tracking

Oct 29, 2018

Prof. Raluca Ada Popa

Slides adapted from Univ of Michigan 583 Fall 12

Announcements

- Exam next Wednesday
  - Open book
  - All lectures except for this one
- Presenter: Pasin
- No writer, but we will post slides

Taint tracking

- A commonly used tool in systems security
- Helps track the flow of data through a program

In a nutshell:

- Data from sensitive sources (e.g., private or potentially malicious sources) is initially tainted
- Other data influenced by this data gets tainted too

Applications

Can be used in a number of ways:

- Unknown Vulnerability Detection
  - E.g. Taint Checking in Ruby and Perl
  - Any data input from an outside user is tainted (e.g., suspicious)
  - If it is used to set another variable, that gets tainted too
  - If a tainted variable gets used directly in a SQL query or a system call, flag as problematic
- Malware Analysis
  - What is the software doing with sensitive data?
  - Ex. TaintDroid
  - Any data from a private source (e.g., text messages) is tainted
  - Track where it is sent

Dynamic Taint Analysis

- Track information flow through a program at runtime

Set a taint tracking policy:

- Identify sources of taint – “TaintSeed”
  - » What are you tracking?
    - Untrusted input
    - Sensitive data
- Taint Policy – “TaintTracker”
  - » Propagation of taint
- Identify taint sinks – “TaintAssert”
  - » Taint checking
    - Special calls
      - Jump statements
    - Outside network

Taint Analysis in Action

Example Policy

Taint seed example: Any input from an untrusted source is tainted

$$\text{Input}\ \frac{t = \text{IsUntrusted}(\mathit{src})}{\text{get\_input}(\mathit{src}) \downarrow t}$$

Taint tracker example: For a binary operation, the taint of the result is the OR of the taints of each operator input

$$\text{BinOp}\ \frac{t_1 = \tau[x_1]\ ,\ t_2 = \tau[x_2]}{x_1 + x_2 \downarrow t_1 \vee t_2}$$

Taint assert: Any goto statement can only go to a nontainted address

$$P_{goto}(t_a) = \neg\, t_a$$

(Must be true to execute)

tainted untainted

x = get_input( )

y = x + 42

…

goto y

Input is tainted

TaintSeed

$$\text{Input}\ \frac{t = \text{IsUntrusted}(src)}{\text{get\_input}(src) \downarrow t}$$

Δ

Var	Val
x	7

τ

Var	Tainted?
x	T

tainted untainted

x = get_input()

y = x + 42

...

goto y

Data derived from user input is tainted

Δ

Var	Val
x	7
y	49

TaintTracker

$$\text{BinOp}\ \frac{t_1 = \tau[x_1]\ ,\ t_2 = \tau[x_2]}{x_1 + x_2 \downarrow t_1 \vee t_2}$$

τ

Var	Tainted?
x	T
y	T

tainted untainted

x = get_input()

y = x + 42

…

goto y

Policy Violation Detected

TaintAssert

$P_{goto}(t_a) = \neg t_a$

(Must be true to execute)

Δ

Var	Val
x	7
y	49

τ

Var	Tainted?
x	T
y	T

x = get_input( )

y = …

…

goto y

Jumping to overwritten return address

<u>Helpful with buffer overflow:</u>

```
…
strcpy(buffer,argv[1]) ;
…
return ;
```

Pay attention to

- False Negatives
  - » Use control flow to change value without gathering taint

    Example: if (x == 0) y=0; else if (x == 1) y=1;
    - Equivalent to x=y;
  - » Tainted index into a hardcoded table

    Policy – value translation is not tainted
  - » Hard to enumerating all sources of taint
- False Positives
  - » Sanity Checks not removing taint

    - Requires fine-tuning
    - *Taint sanitization problem*
    - *Usually many and a lot of taint!*

How about loading from memory?

Memory Load

Variables

Δ

Var	Val
x	7

τ

Var	Tainted?
x	T

Memory

μ

Addr	Val
7	42

τ_μ

Addr	Tainted?
7	F

Problem: Memory Addresses

```
x = get_input(  )
y = load( x )
...
goto y
```

All values derived from user input are tainted??

Δ

Var	Val
x	7

μ

Addr	Val
7	42

τ_μ

Addr	Tainted?
7	F

Policy 1: Taint depends only on the memory cell

x = get_in

y = load(

...

goto y

Undertainting

Failing to identify tainted values

- e.g., missing exploits

Δ

Var	Val
x	7

Addr	Val
7	42

Taint Propagation

$$\text{Load}\ \frac{v = \Delta[x]\ ,\ t = \tau_\mu[v]}{\text{load}(x) \downarrow t}$$

τ_μ

Addr	Tainted?
7	F

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

x = get_i

y = load(

...

goto y

Overtainting

Unaffected values are tainted

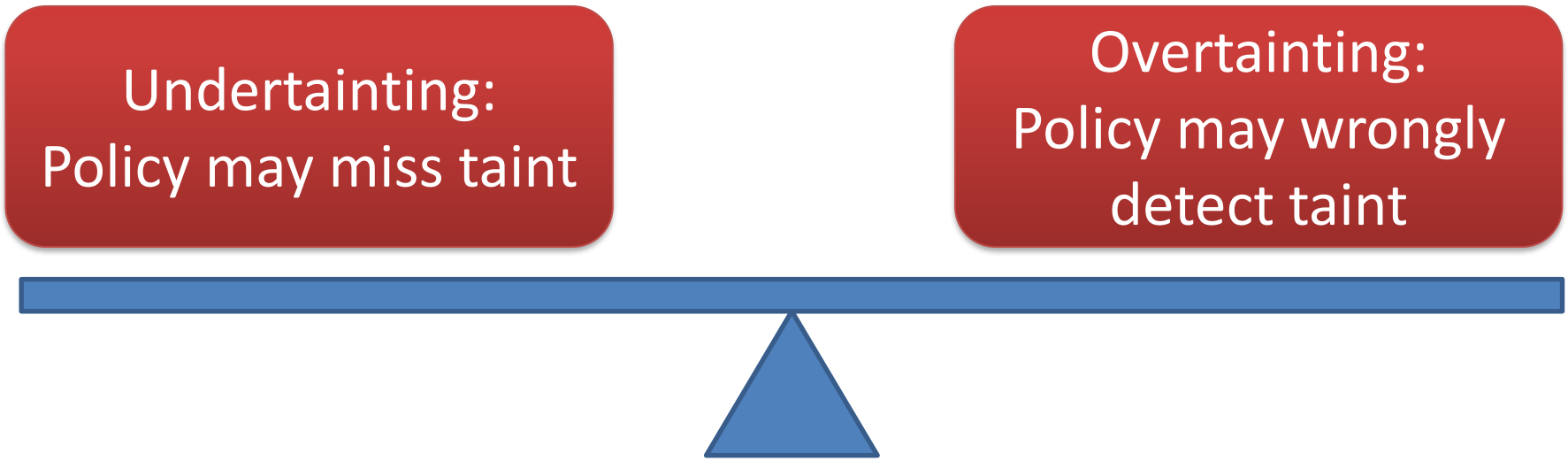
- e.g., flag exploits on safe inputs

Memory
Address expression is tainted
printa
printb

Taint Propagation

$$\text{Load}\ \frac{v = \Delta[x]\ ,\ t = \tau_\mu[v],\ t_a = \tau[x]}{\text{load}(x) \downarrow t \vee t_a}$$

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



Summary

- Taint tracking can be used to track flow of private data or suspicious inputs
- Further reading: *All You Ever Wanted to Know About Dynamic Taint Analysis,* Schwartz et al, Oakland 2010
- Next up: Pasin on TaintDroid