Taint tracking

Suman Jana

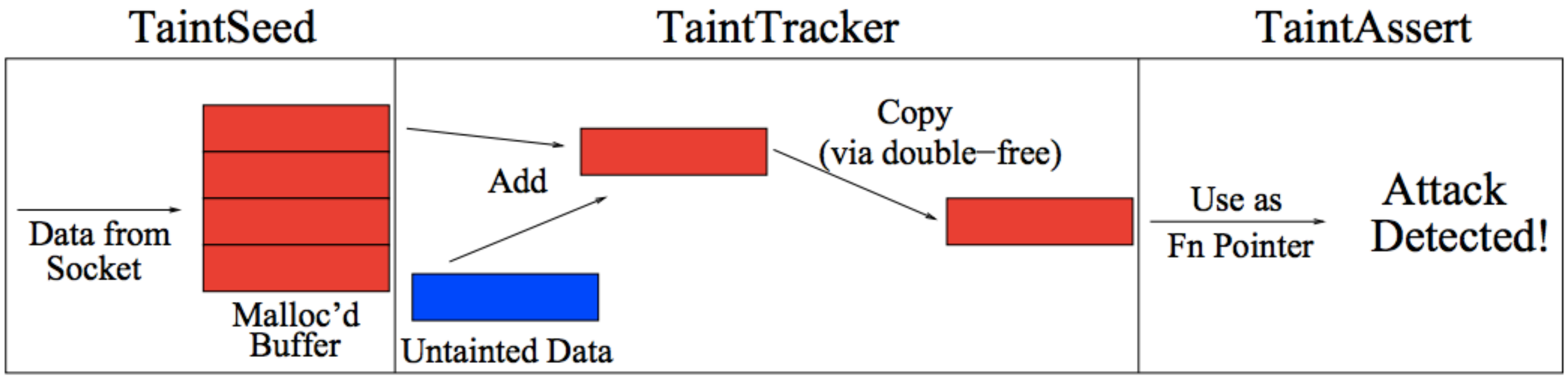
Dynamic Taint Analysis

- Track information flow through a program at runtime
- 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, Format strings, etc.
    - Outside network

TaintCheck (Newsome et al.)

- Performed on x86 binary
  - No need for source
- Implemented using Valgrind skin
  - X86 -> Valgrind's Ucode
  - Taint instrumentation added
  - Ucode -> x86
- Sources -> TaintSeed
- Taint Policy -> TaintTracker
- Sinks -> TaintAssert
- Add on "Exploit Analyzer"

TaintCheck (Newsome et al.)



- TaintSeed: Mark untrusted data as tainted
- TaintTracker: Track each instruction, propagate taint
- TaintAssert: Check is tainted data is used dangerously

TaintSeed

- Marks any data from untrusted sources as "tainted"
  - Each byte of memory has a four-byte shadow memory that stores a pointer to a Taint data structure if that location is tainted
  - Else store a NULL pointer

TaintTracker

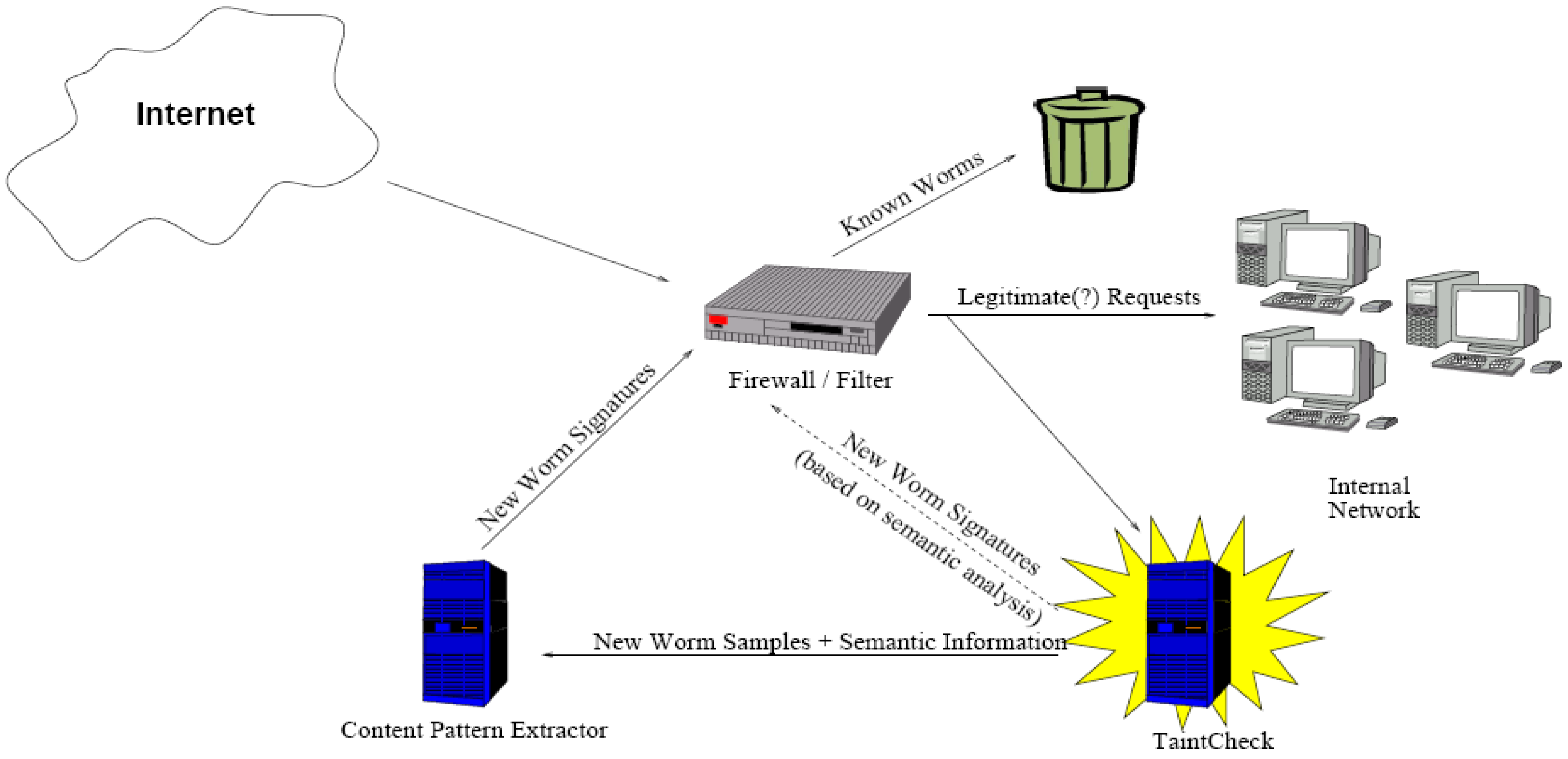
- Tracks each instruction that manipulates data in order to determine whether the result is tainted.
  - When the result of an instruction is tainted by one of the operands, TaintTracker sets the shadow memory of the result to point to the same Taint data structure as the tainted operand.

TaintAssert & Exploit Analyzer

- TaintAssert
  - Checks whether tainted data is used in ways that its policy defines as illegitimate
- Exploit Analyzer
  - Backtrace chain of taint structures: provides useful information about how the exploit happened, and what the exploit attempts to do
  - Useful to generate exploit fingerprints
  - Transfer control to sandbox for analysis

Automatic Signature Generation

- Find value used to override return address – typically fixed value in the exploit code


Internet
Known Worms
Legitimate(?) Requests
Firewall / Filter
Internal Network
New Worm Signatures
New Worm Signatures (based on semantic analysis)
New Worm Samples + Semantic Information
Content Pattern Extractor
TaintCheck

Taint Analysis in Action

🔴 tainted 🔵 untainted

```
➡ x = get_input( )
  y = x + 42
  …
  goto y
```

Input is tainted

TaintSeed

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

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

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

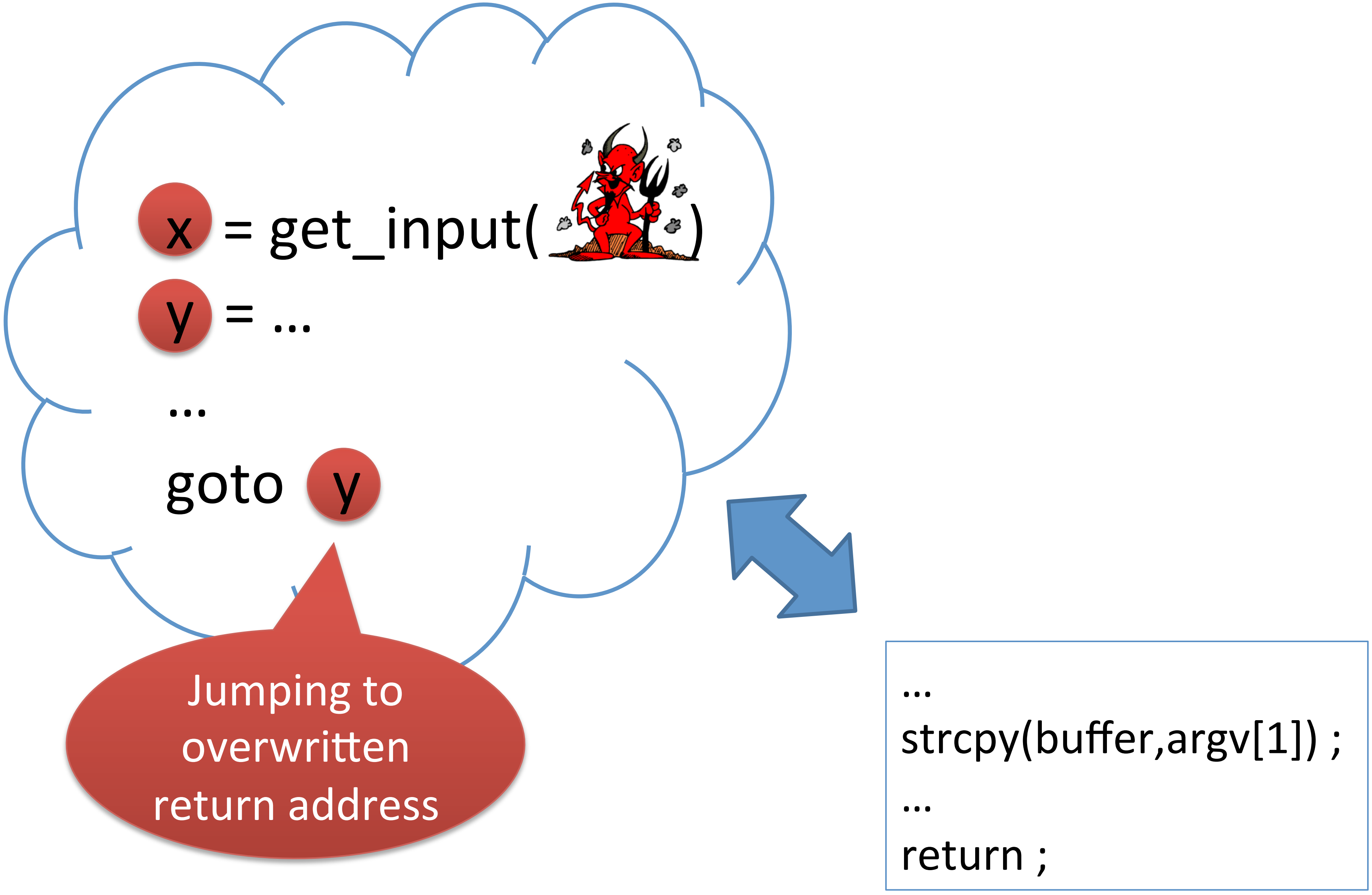
(Must be true to execute)

Δ

Var	Val
x	7
y	49

τ

Var	Tainted
x	T
y	T



```
x = get_input(   )
y = …
…
goto y
```

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

Policy Considerations?

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/T?

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_i

y = load(

...

goto y

Δ

Var	Val
x	7

Addr	Val
7	42

Undertainting

Failing to identify tainted values

- e.g., missing exploits

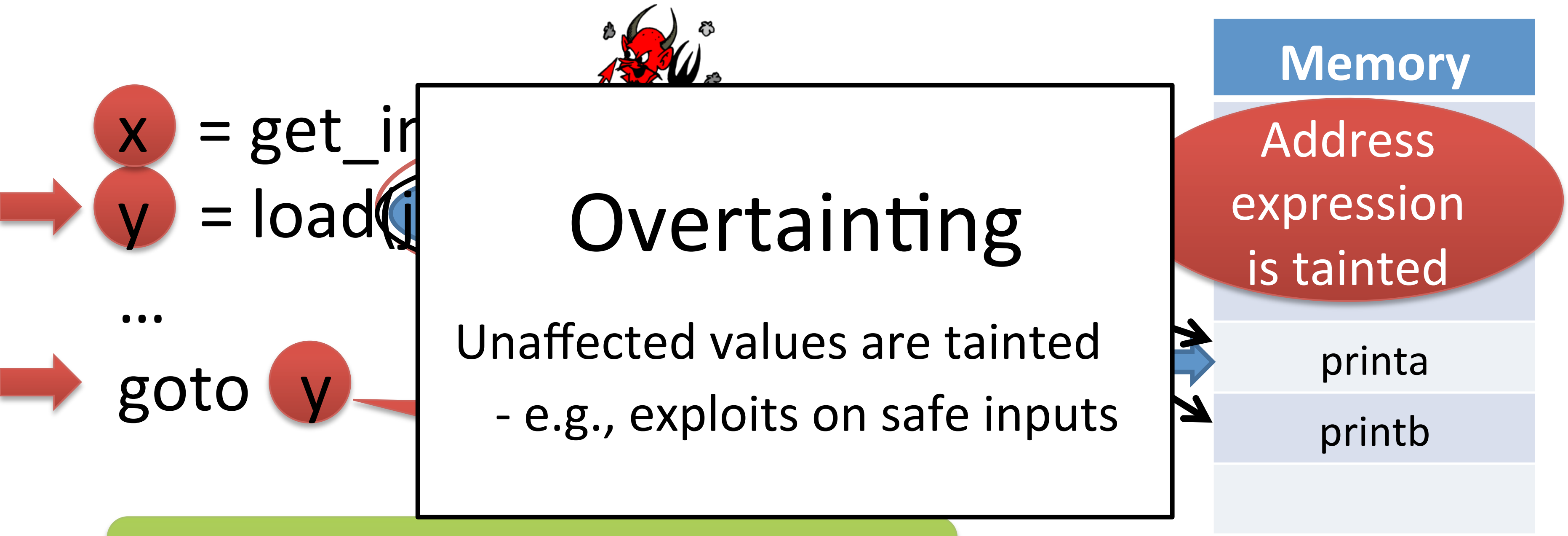
Taint Propagation

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

τ_μ

Addr	Tainted
7	F

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



Overtainting

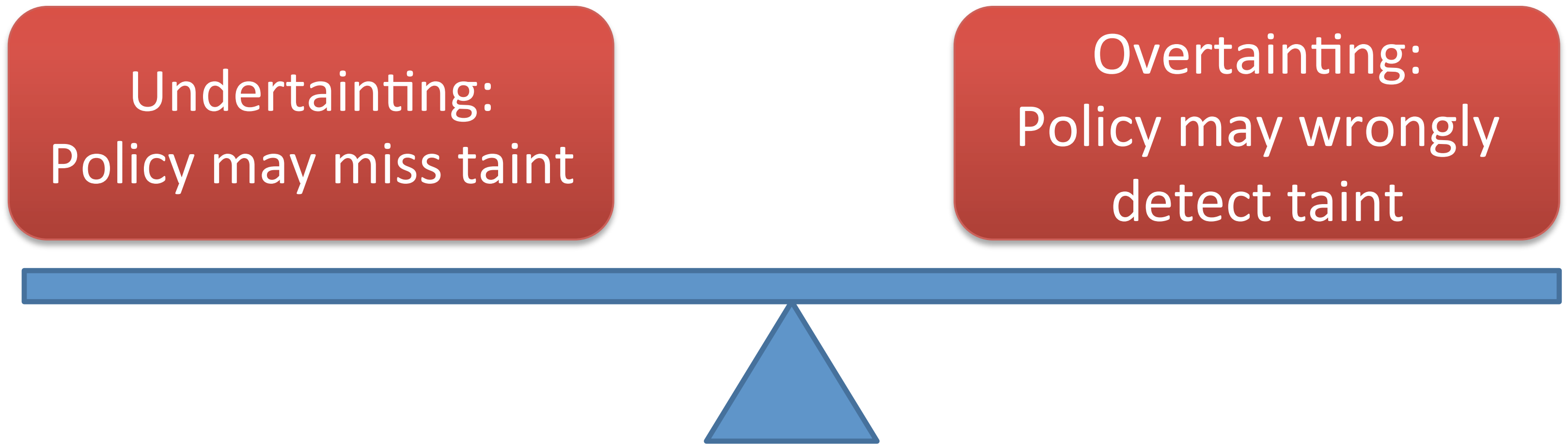
Unaffected values are tainted

- e.g., exploits on safe inputs

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



TaintCheck Evaluation

Effectiveness of TaintCheck

- 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
  - Enumerating all sources of taint

- False Positives
  - Vulnerable code?
  - Sanity Checks not removing taint
    - Requires fine-tuning
    - Taint sanitization problem

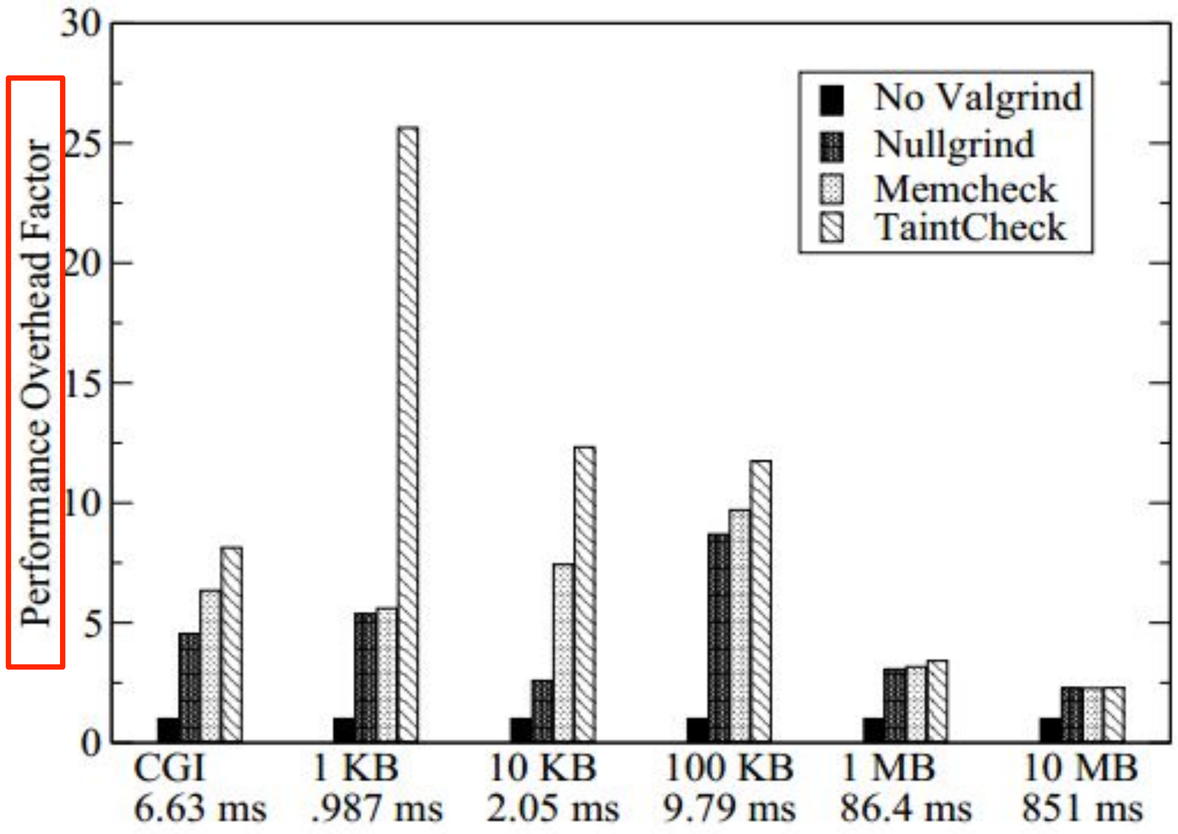
Effectiveness of TaintCheck

- Does TaintCheck raise false alerts for existing code?
  - network programs: apache, ATPhttpd, bftpd, cfingerd, and named
  - client programs: ssh and firebird
  - non-network programs: gcc, ls, bzip2, make, latex, vim, emacs, and bash
- Networked programs: 158K+ DNS queries
  - No false +ves
- All client and non-network programs (tainted data is stdin):
  - Only vim and firebird caused false +ves (data from config files used as offset to jump address)

TaintCheck - Attack Detection

- Synthetic Exploits
  - Buffer overflow -> function pointer
  - Buffer overflow -> format string
  - Format string -> info leak
- Actual Exploits
  - 3 real world examples

TaintCheck Performance



Performance overhead for Apache