ECE 382N-Sec (FA25):

L12: Information-Flow Tracking in Hardware

Neil Zhao

neil.zhao@utexas.edu

Recap: Speculative Taint Tracking (STT)

Tainted

Root of taint

```
void vulnerable_code(size_t idx) {
  if (idx < array_size) { // B1
    u8 data = array[idx]; // speculative access, execute & taint!
    u8 X = data + 2; // not a transmitter, execute!
    if (idx % 2) { // B2
      u8 junk = glb_array[X*4096]; // transmitter, delay!
    }
  }
}
```

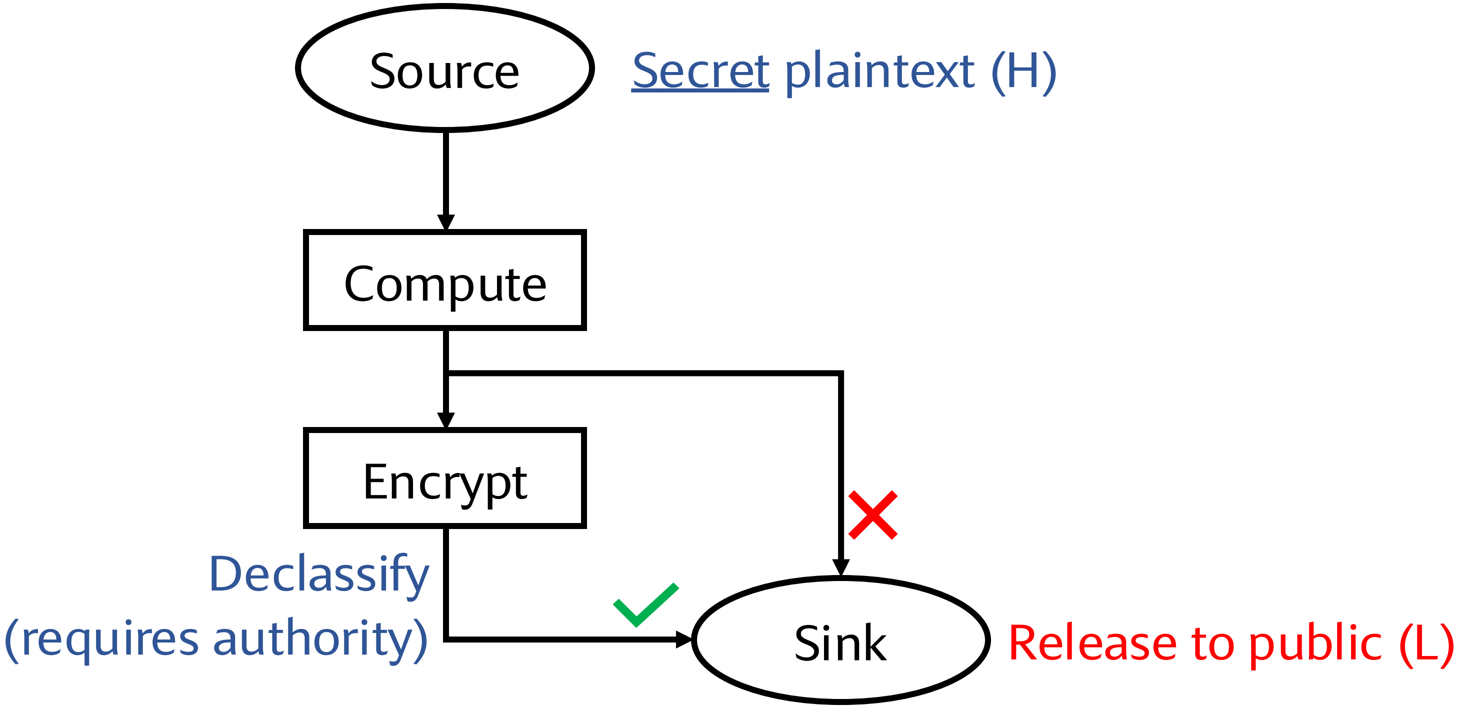
Recap: Speculative Taint Tracking (STT)

Leakage through implicit channels

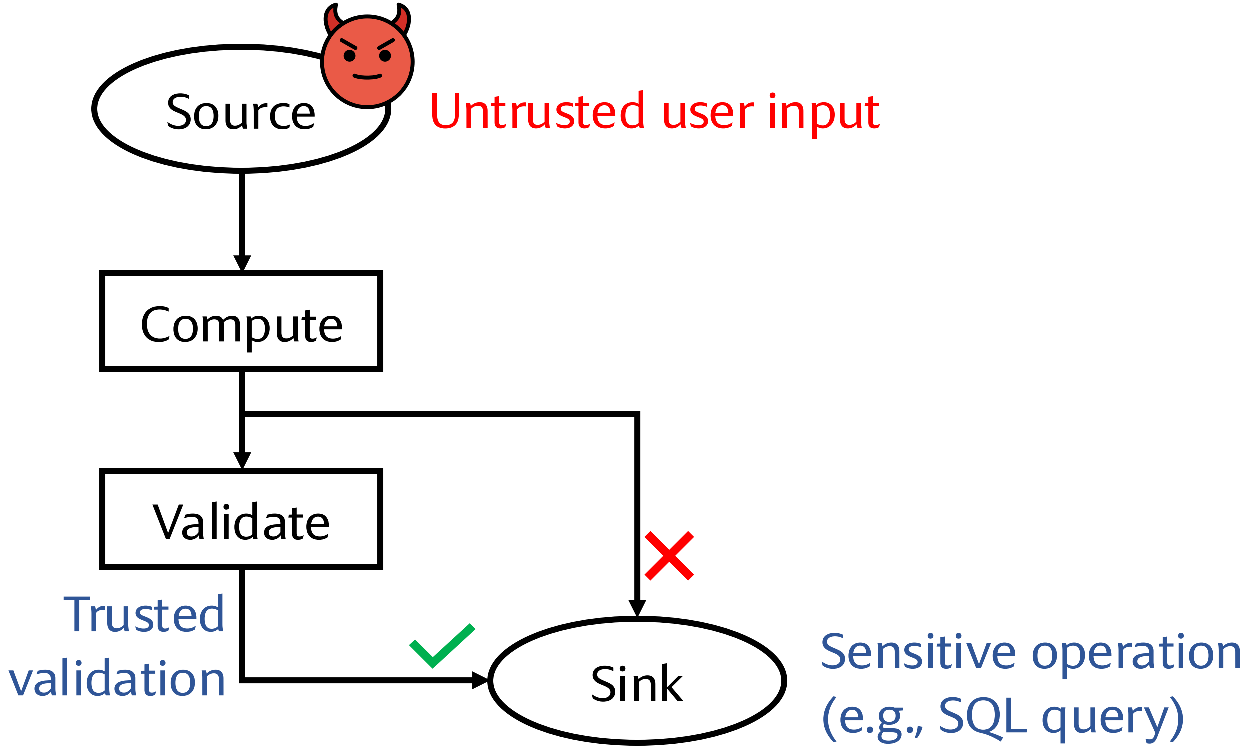
Tainted

```
void vulnerable_code(size_t idx) {
  if (idx < array_size) {
    u8 data = array[idx]; // speculative access
    if (data & 1) { // B1
      div();
    } else {
      mul();
    }
  }
}
```

Information Flow for Confidentiality



Information Flow for Integrity



Explicit Information Flow

```
(H) R1 = load ptr_to_secret;
(H) R2 = R1 + 10;
(H) R3 = R2 & R1;
    store R3 → addr1;

??? R4 = load addr2;
```

Conservatively taint the output if either inputs is tainted

Can depend on runtime states. Difficult for static analysis to handle

Implicit Information Flow

```
(H) if (secret) {
(H)   R1 = 10;
    } else {
(H)   R1 = 20;
    }
```

```
(H) R1 = secret;
    store 10 → R1;
```

Can taint the entire address space!

Taint explosion

Complete Information Flow Tracking from the Gates Up*

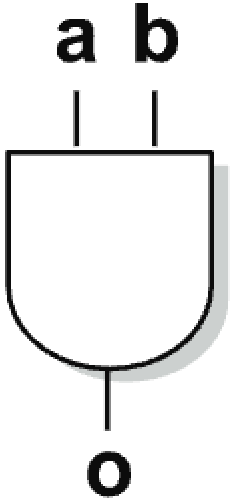
Complete Information Flow Tracking from the Gates Up

Mohit Tiwari Hassan M G Wassel Bita Mazloom Shashidhar Mysore Frederic T Chong
Timothy Sherwood

Department of Computer Science, University of California, Santa Barbara
{tiwari,hwassel,betamaz,shashimc,chong,sherwood}@cs.ucsb.edu

*Tiwari et al., “Complete Information Flow Tracking from the Gates Up,” ASPLOS ’09

A Sound and Precise Taint Propagation Policy for AND Gates



Logic Truth Table

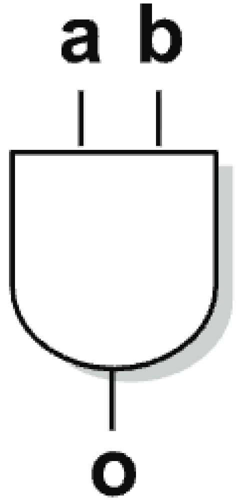
a	b	out
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Tainted b and untainted a

a	b	a_t	b_t	out_t
0	0	0	1	**0**
0	1	0	1	**0**
1	0	0	1	**1**
1	1	0	1	**1**

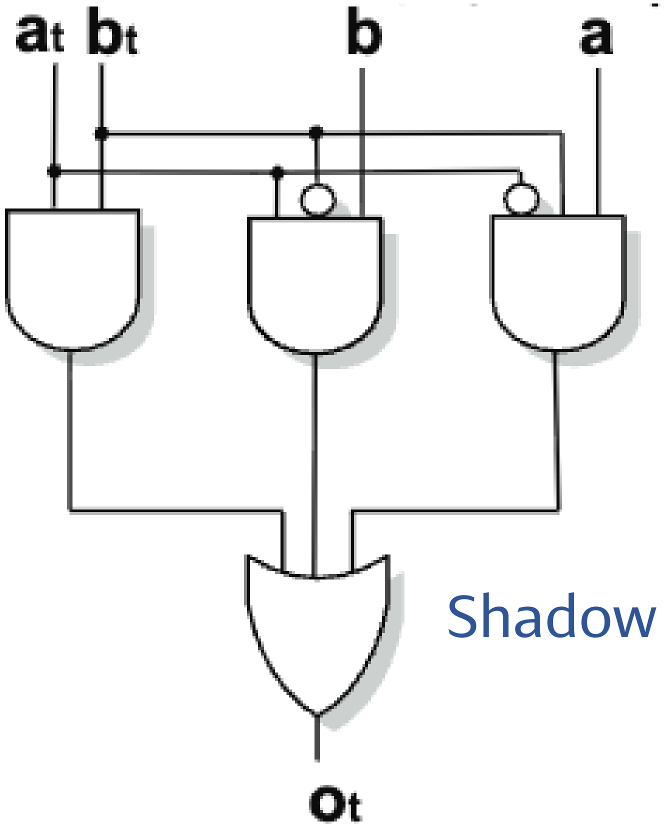
*Tiwari et al., "Complete Information Flow Tracking from the Gates Up," ASPLOS '09

A Sound and Precise Taint Propagation Policy for AND Gates



Logic Truth Table

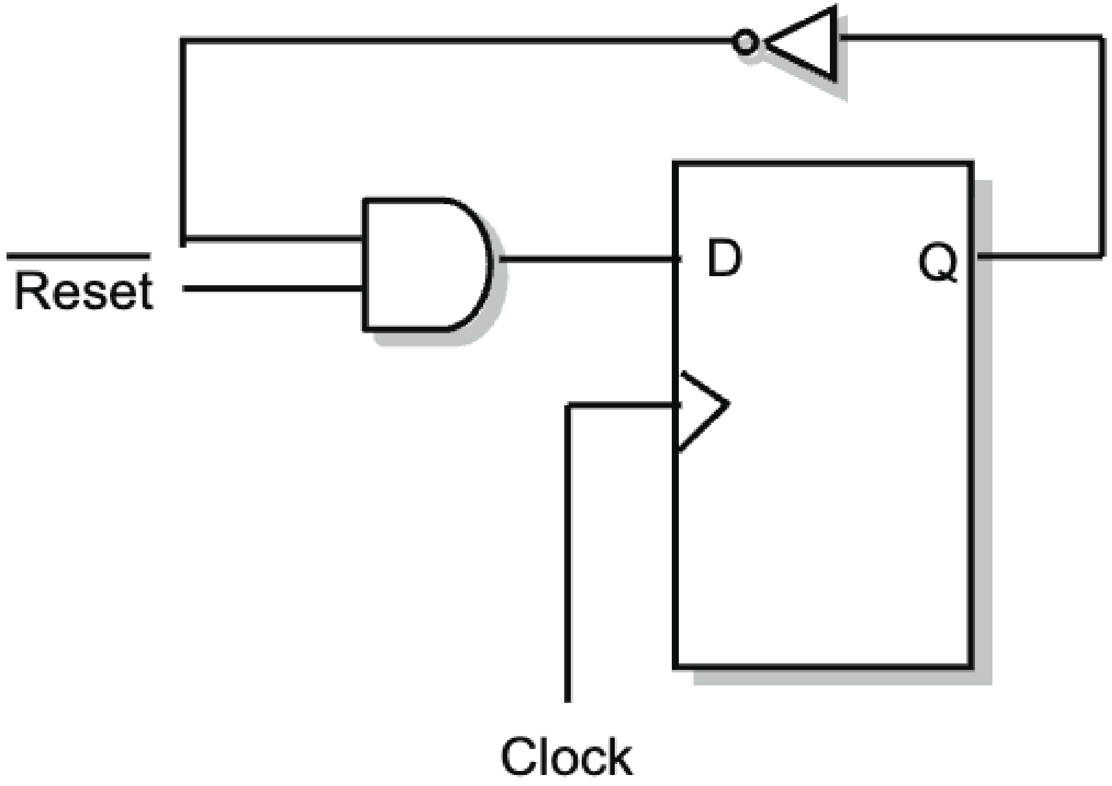
a	b	out
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**



Shadow AND gate

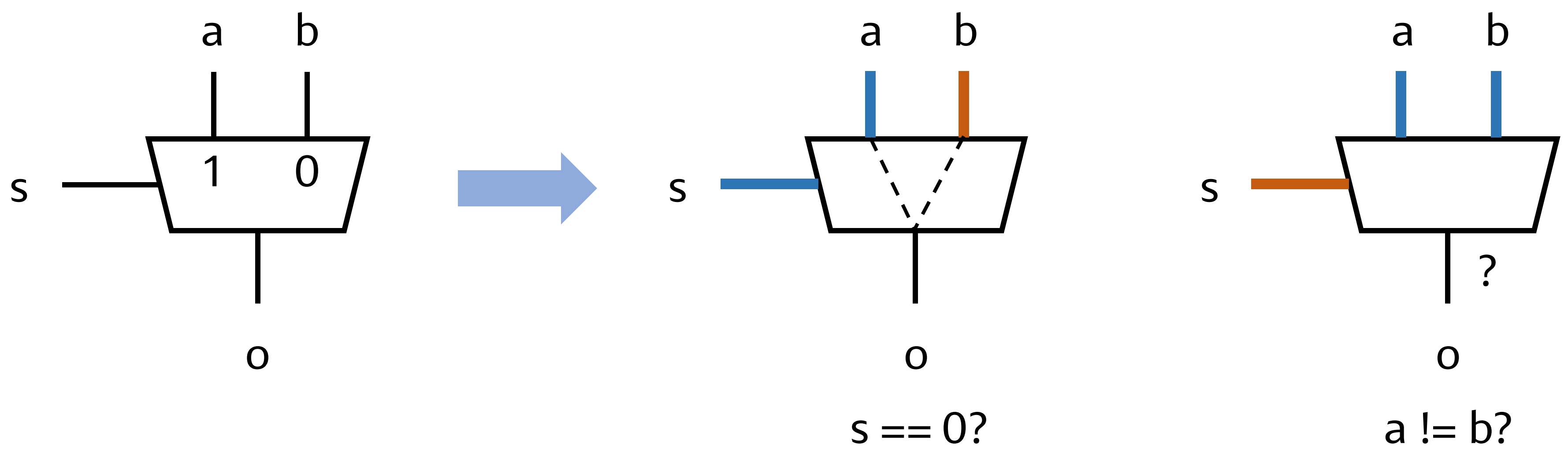
*Tiwari et al., "Complete Information Flow Tracking from the Gates Up," ASPLOS '09

A One-Bit Counter



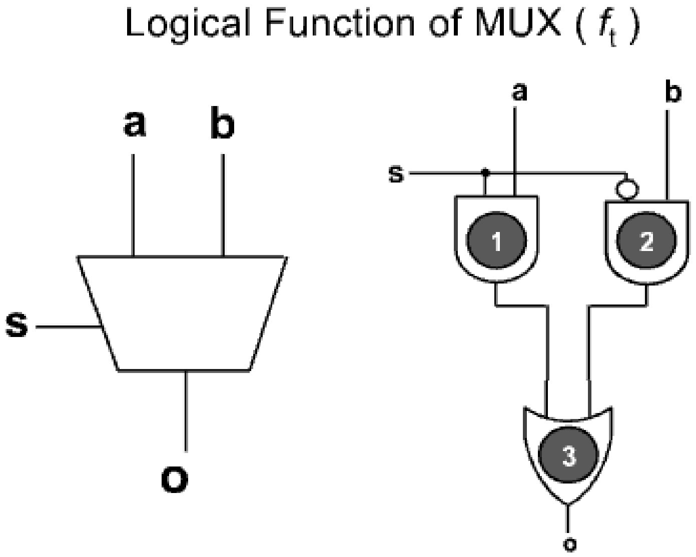
*Tiwari et al., "Complete Information Flow Tracking from the Gates Up," ASPLOS '09

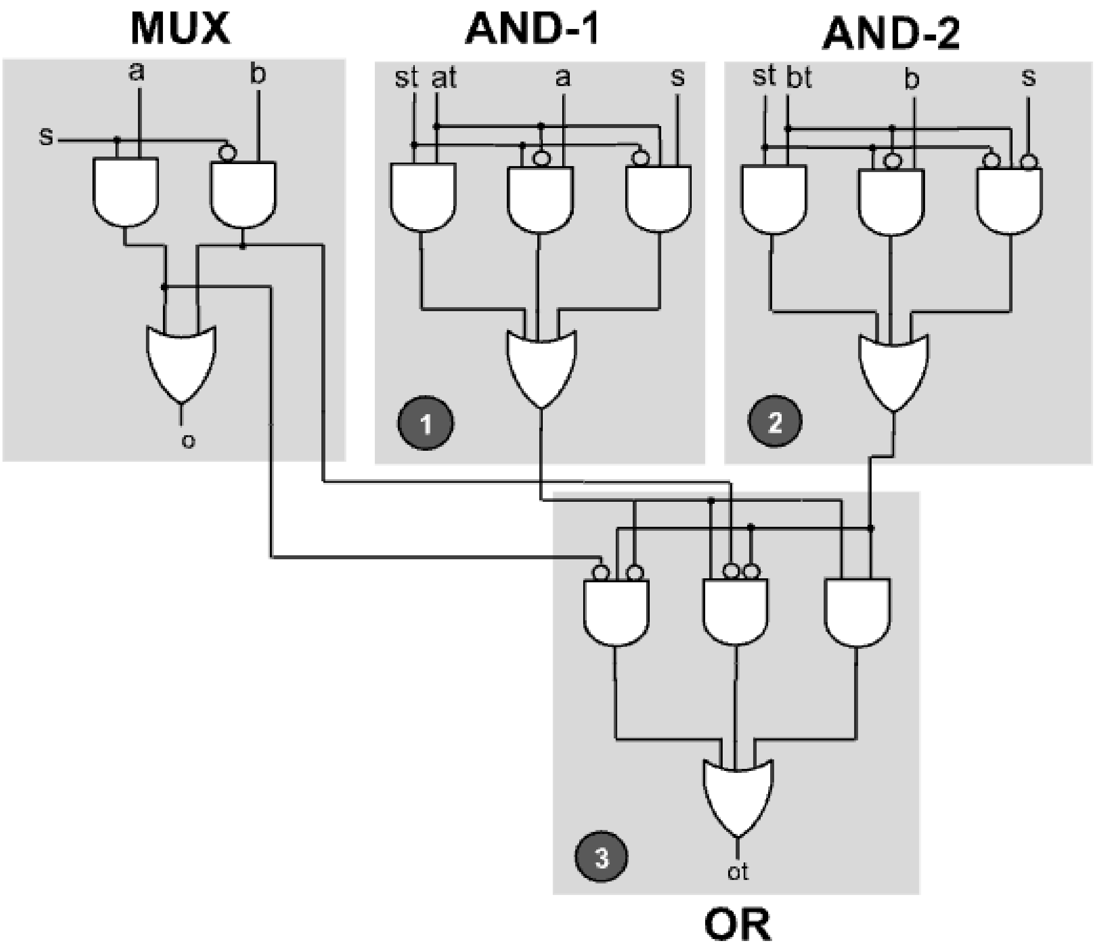
Multiplexer



*Tiwari et al., "Complete Information Flow Tracking from the Gates Up," ASPLOS '09

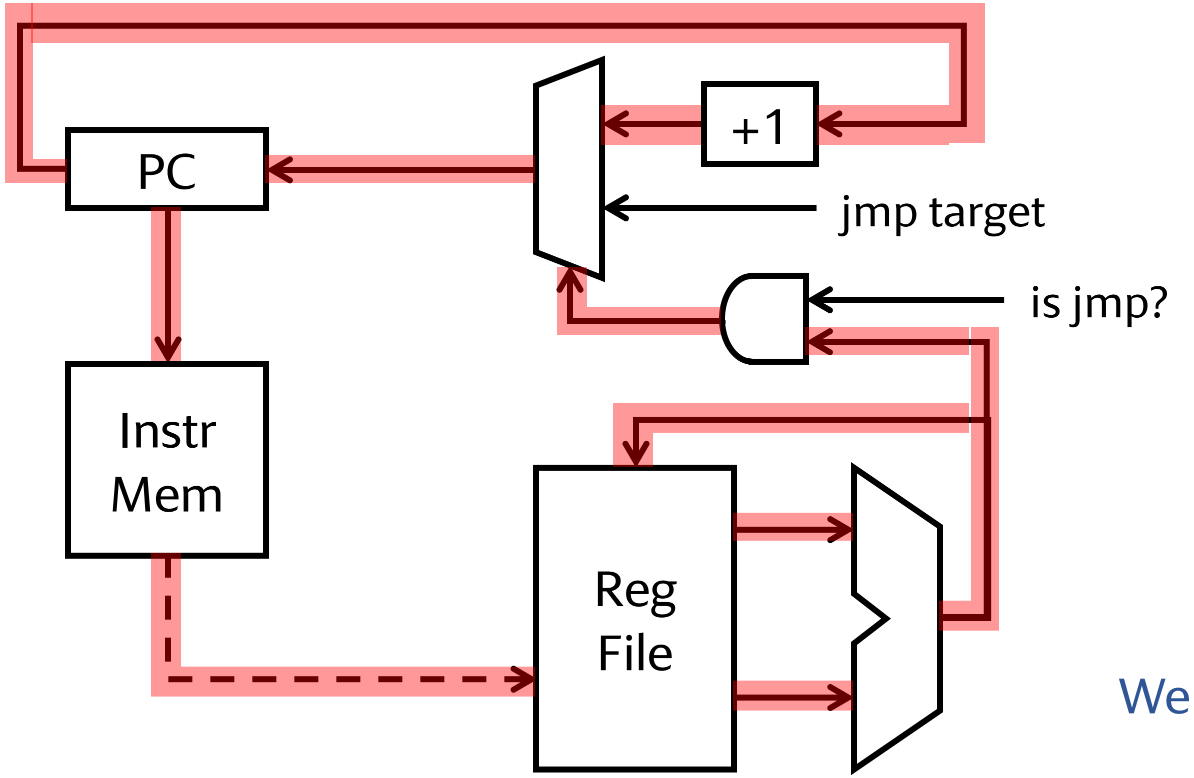
Multiplexer





*Tiwari et al., "Complete Information Flow Tracking from the Gates Up," ASPLOS '09

Prevent Taint Explosion: Covert Implicit Flow into Explicit Flow

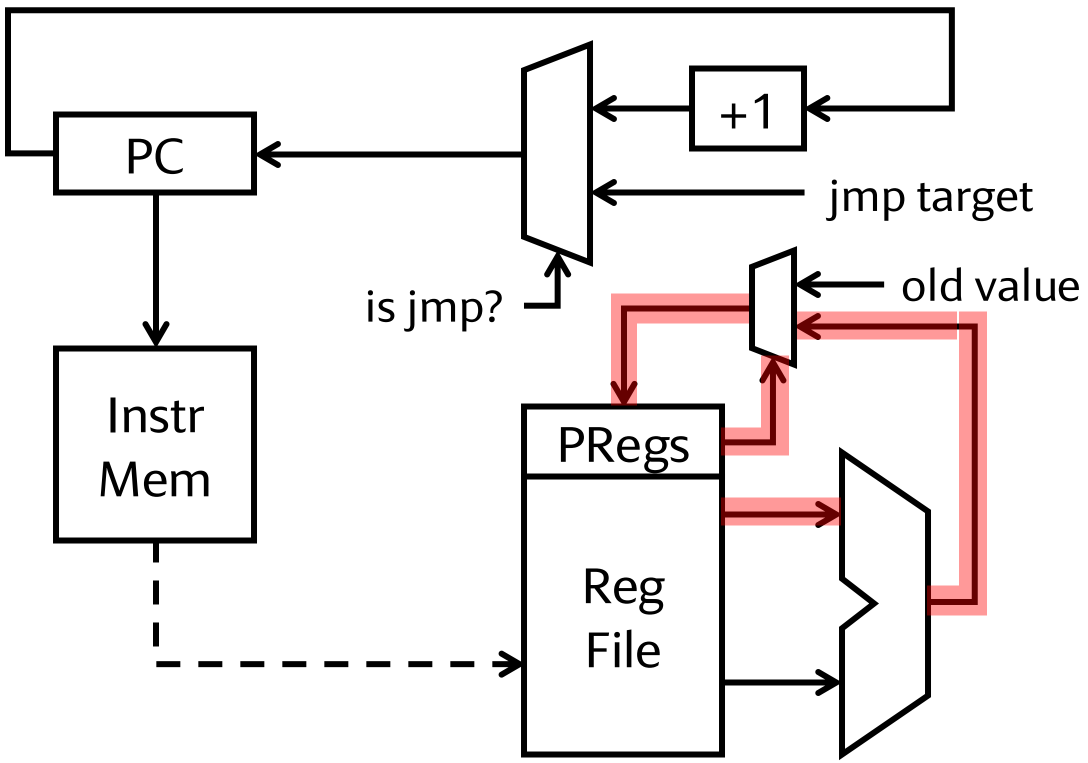



```
0x1: if X == 0; jmp 0x3
0x2: R1 = 10;
0x3: R2 = 10;
```

We want to avoid tainting the PC

*Tiwari et al., "Complete Information Flow Tracking from the Gates Up," ASPLOS '09

Prevent Taint Explosion: Covert Implicit Flow into Explicit Flow




```
0x1: P1 = (X == 0);
0x2: (P1) R1 = 10;
0x3: R2 = 10;
```

*Tiwari et al., "Complete Information Flow Tracking from the Gates Up," ASPLOS '09

Prevent Taint Explosion: Covert Implicit Flow into Explicit Flow

Bounding Loops

```
0x1: ...
...
0xa: countjump (0x1), 100
```

The number of loop iterations is statically determined

"countjump" cannot be predicated

Constraining Loads/Stores

```
0x1: init-counter C0
0x2: R1 = load 0x200 + C0
0x3: inc-counter C0
0x4: inc-counter C0
...
0xa: countjump (0x2), 100
```

Because the PC cannot be tainted, the addresses are guaranteed to be untainted

*Tiwari et al., "Complete Information Flow Tracking from the Gates Up," ASPLOS '09

Putting All Pieces Together

Summary:

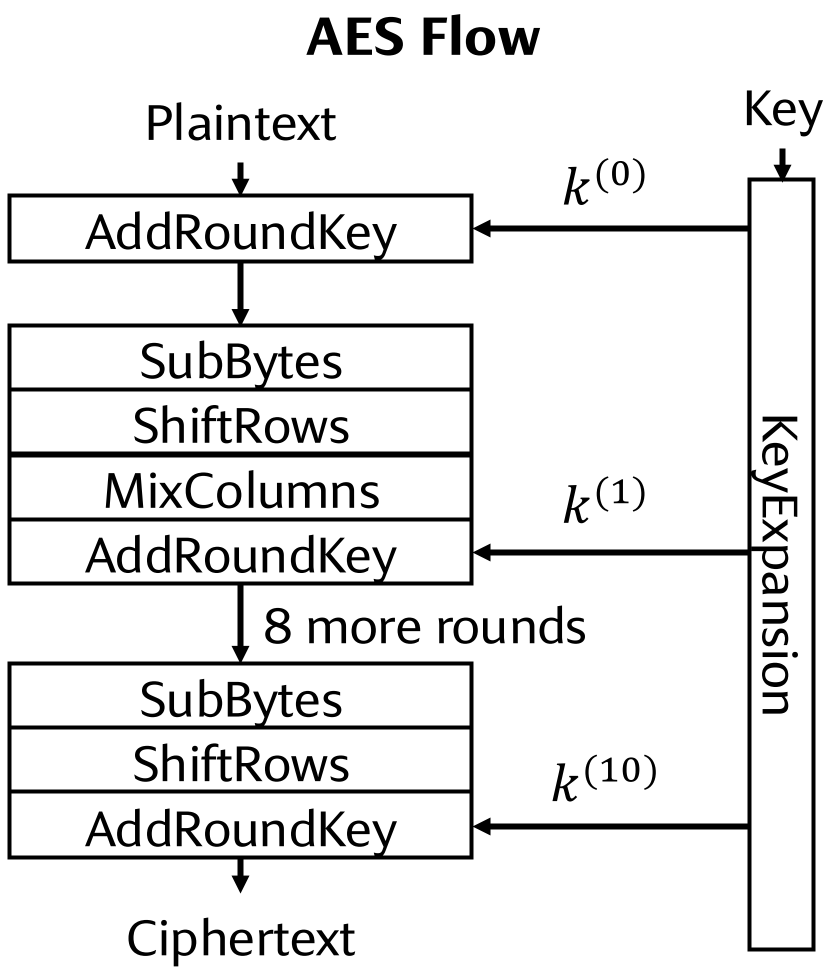
- ISA-level restriction that prevents programmer from writing code that can potentially lead to taint explosion
- An FPGA prototype (information-flow tracking at the bit granularity)

- Good for security-sensitive programs

- Area overhead (+70%)
- Performance overhead
- Limited programmability

*Tiwari et al., "Complete Information Flow Tracking from the Gates Up," ASPLOS '09

Speculative Privacy Tracking (SPT)




```
for (size_t r = 1; r <= 10; r++) {
  eax, …, edx = AES_Round(eax, …, edx, rnd_key);
}

if (/* false */) {
  transmit(eax); // transmitter
}
```

Not protected by STT!

*Choudhary et al., "Speculative Privacy Tracking (SPT): Leaking Information From Speculative Execution Without Compromising Privacy," MICRO '21

Key Idea of SPT: Leaked in Non-Spec Exec → Not a Secret

```
v1, v2 = ...;
transmit(v1);
if (/* false */) {
  transmit(v1);
  transmit(v1 + 10);
  transmit(v1 + v2);
}
```

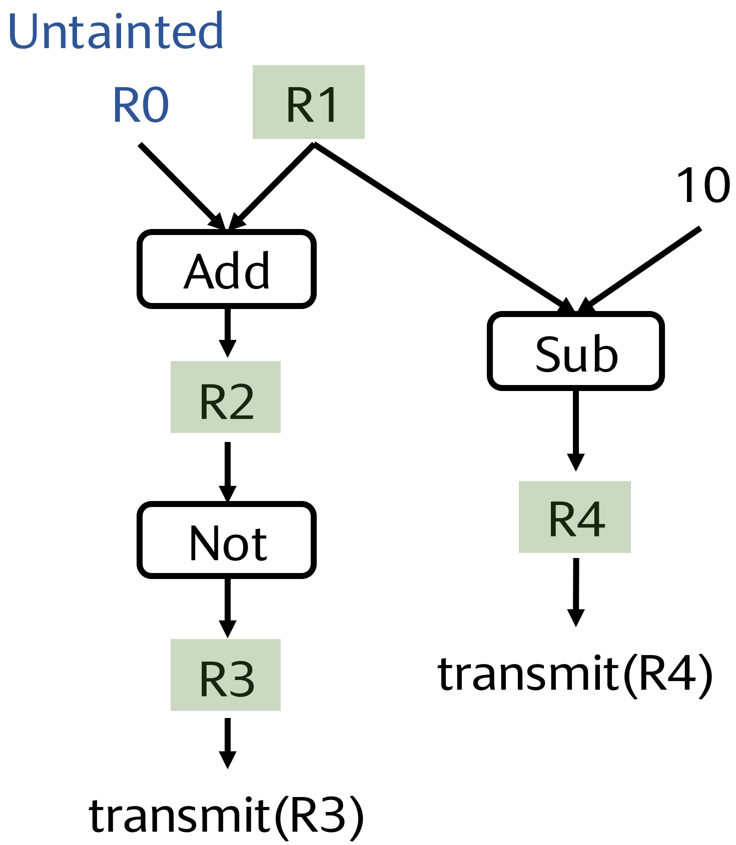
Everything is a secret until it's leaked/declassified by
a non-speculative transmitter

*Choudhary et al., "Speculative Privacy Tracking (SPT): Leaking Information From Speculative Execution Without Compromising Privacy," MICRO '21

Backward Untaint Propagation

```
R0, R1 = ...;
R2 = R0 + R1;
R3 = ~R2;
transmit(R3);
if (/* false */) {
  R4 = R1 - 10;
  transmit(R4); ✓
}
```

This example assumes that the program and PC are public



*Choudhary et al., "Speculative Privacy Tracking (SPT): Leaking Information From Speculative Execution Without Compromising Privacy," MICRO '21

Blocking Implicit Channels

Similar to STT, if a branch's predicate is tainted:

- Delay branch resolution
- Delay predictor updates

⇒ PC is not tainted

*Choudhary et al., "Speculative Privacy Tracking (SPT): Leaking Information From Speculative Execution Without Compromising Privacy," MICRO '21

Untaint Propagation Through Memory

```
store R1 → addr1;
store R2 → addr2;
store R3 → addr3;

R4 = load addr;
```

```
if (alias(addr, addr3, …))
  R4 = R3;
else if (alias(addr, addr2, …))
  R4 = R2;
else if (alias(addr, addr1, …))
  R4 = R1;
else
  R4 = load addr;
```

Whether "`load addr`" issues leaks addr1, addr2, and addr3

*Choudhary et al., "Speculative Privacy Tracking (SPT): Leaking Information From Speculative Execution Without Compromising Privacy," MICRO '21

Untaint Propagation Through Memory

```
store R1 → addr1;
store R2 → addr2;
store R3 → addr3;

R4 = load addr;
```

```
Tmp = load addr;

if (alias(addr, addr3, …))
  R4 = R3;
else if (alias(addr, addr2, …))
  R4 = R2;
else if (alias(addr, addr1, …))
  R4 = R1;
else
  R4 = Tmp;
```

*Choudhary et al., “Speculative Privacy Tracking (SPT): Leaking Information From Speculative Execution Without Compromising Privacy,” MICRO ’21

Untaint Propagation Through Memory

Untainted Tainted

```
store R1 → addr1;
store R2 → addr2;
store R3 → addr3;

R4 = load addr;

transmit(R4);
```

Leaks tainted addr2 and addr3

```
Tmp = load addr;

if (alias(addr, addr3, …))
  R4 = R3;
else if (alias(addr, addr2, …))
  R4 = R2;
else if (alias(addr, addr1, …))
  R4 = R1;
else
  R4 = Tmp;
```

*Choudhary et al., "Speculative Privacy Tracking (SPT): Leaking Information From Speculative Execution Without Compromising Privacy," MICRO '21

Solution: Delay Untaint Propagation in STL Forwarding

Untainted

```
store R1 → addr1;
store R2 → addr2;
store R3 → addr3;

R4 = load addr;

transmit(R4); ✓
```

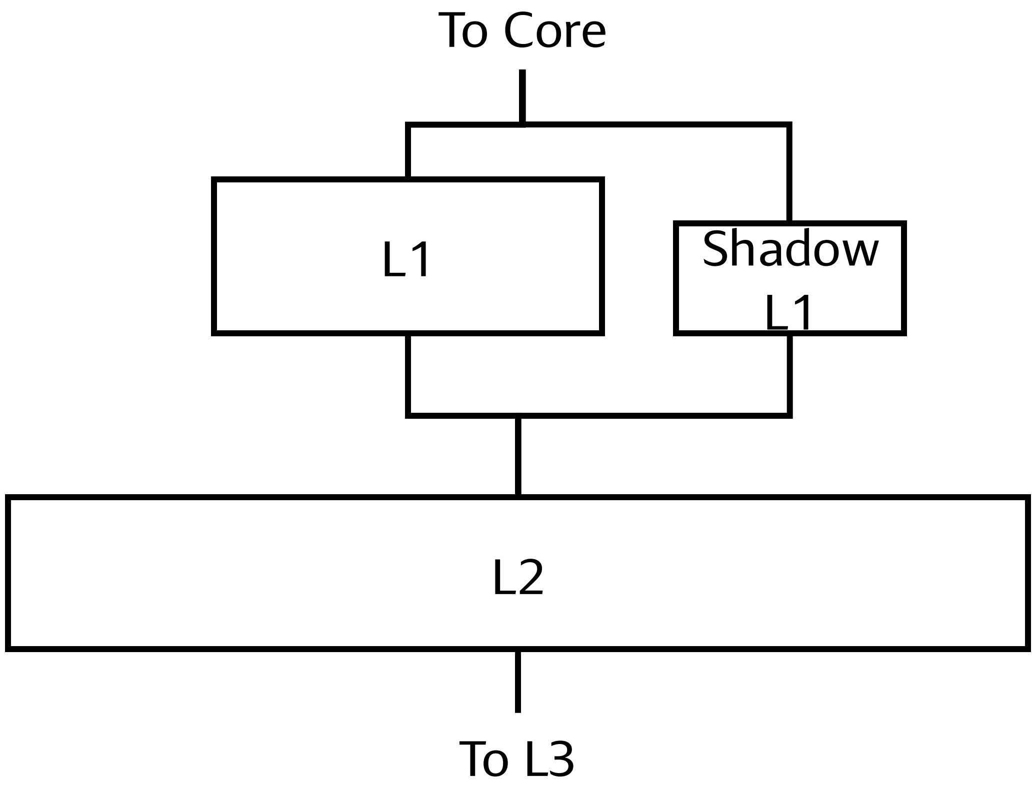
```
Tmp = load addr;

if (alias(addr, addr3, …))
  R4 = R3;
else if (alias(addr, addr2, …))
  R4 = R2;
else if (alias(addr, addr1, …))
  R4 = R1;
else
  R4 = Tmp;
```

Untainted

*Choudhary et al., “Speculative Privacy Tracking (SPT): Leaking Information From Speculative Execution Without Compromising Privacy,” MICRO ’21

Untaint Propagation Through Memory (Shadow L1D)



Data are conservatively tainted when brought into the L1

*Choudhary et al., "Speculative Privacy Tracking (SPT): Leaking Information From Speculative Execution Without Compromising Privacy," MICRO '21

Performance Overhead

- gem5 evaluation
- Futuristic model (consider all forms of speculation)
    - 3.6× overhead → 45%

- Spectre model (consider only control-flow speculation)
    - 3× overhead → 11%
    - STT: 8%

*Choudhary et al., "Speculative Privacy Tracking (SPT): Leaking Information From Speculative Execution Without Compromising Privacy," MICRO '21

Some Questions for Discussion

- [GLIFT] How does GLIFT differ from other tagging/tainting methods discussed earlier in the course, like STT?
- [GLIFT] Could GLIFT serve as the basis for “verifiable secure enclaves” where secrets are provably isolated at the hardware level?
- [GLIFT] Microarchitectural optimization?
- [SPT] Often times paper's like these employ a hardware only or software only solution, if the authors opened up the ability to co-design with software, what kind of solution would SPT now look like?
(+ a similar question)
- [SPT] Is there ever any reason to use STT over SPT?
- [SPT] The paper shows 45% overhead in the Futuristic model but only 11% in the Spectre model. Why such a large difference? Which model is more realistic for defending against real-world attacks?