Taint Analysis

Basic Idea:
Keep track of values derived from user input

Taint Analysis

- is a data flow analysis
- Define 3 things: Taint Introduction + Taint propagation + Taint enforcement policy
- Taint Introduction: What is input?
- Taint Propagation: “Any value derived from tainted data is data”
- Taint Enforcement: “Users should not be able to determine jump target addresses”
  - E.g., overwriting the return address means a user has overwritten a jump target (the return) address

Example

```
i = get_input();
two = 2;
if(i %2 == 0){
    j = i+two;
    l = k;
} else {
    k = two*two;
    l = k;
}
jmp l;
```

Two Approaches

- Static Approach: Look at program text
- Dynamic Approach: Look at instructions executed in program run

Dynamic Taint Analysis

Analysis performed at each step of execution of a single run

Example

```
→i = get_input();
  two = 2;
  if(i %2 == 0){
      j = i+two;
      l = k;
  } else {
      k = two*two;
      l = k;
  }
  jmp l;
```

Variable	Value (Int)	Taint Status (T/F)
i	6	T

$$\frac{}{\text{get\_input} \Downarrow T}\ \text{INPUT}$$

Example

```
i = get_input();
two = 2;
if(i %2 == 0){
    j = i+two;
    l = k;
} else {
    k = two*two;
    l = k;
}
jmp l;
```

Variable	Value (Int)	Taint Status (T/F)
i	6	T
two	2	F

$$\frac{n \text{ is a constant}}{n \Downarrow F}\ \text{CONST}$$

Example

```
i = get_input();
two = 2;
if(i %2 == 0){
    j = i+two;
    l = k;
} else {
    k = two*two;
    l = k;
}
jmp l;
```

Variable	Value (Int)	Taint Status (T/F)
i	6	T
two	2	F

Example

```
i = get_input();
two = 2;
if(i %2 == 0){
    j = i+two;
    l = k;
} else {
    k = two*two;
    l = k;
}
jmp l;
```

Variable	Value (Int)	Taint Status (T/F)
i	6	T
two	2	F
j	8	T

$$\frac{t_1 = \text{taint of } x_1 \quad t_2 = \text{taint of } x_2 \quad t = t_1 \vee t_2}{x_1 \square x_2 \Downarrow t}\ \text{OP}$$

"Anything derived from tainted data is tainted."

Example

```
i = get_input();
two = 2;
if(i %2 == 0){
    j = i+two;
    l = j;
} else {
    k = two*two;
    l = k;
}
jmp l;
```

Variable	Value (Int)	Taint Status (T/F)
i	6	T
two	2	F
j	8	T
l	8	T

Example

```
i = get_input();
two = 2;
if(i %2 == 0){
    j = i+two;
    l = k;
} else {
    k = two*two;
    l = k;
}
jmp l;
```

Variable	Value (Int)	Taint Status (T/F)
i	6	T
two	2	F
j	8	T
l	8	T

$$\frac{\text{TAINT STATUS OF X IS } f}{\text{jmp } x \text{ OK}}\ \text{JMP}$$

"Users input should not be used as jump target"

Example

```
i = get_input();
two = 2;
if(i %2 == 0){
    j = i+two;
    l = k;
} else {
    k = two*two;
    l = k;
}
jmp l;
```

Variable	Value (Int)	Taint Status (T/F)
i	6	T
two	2	F
j	8	T
l	8	T

"Users input should not be used as jump target"

Dynamic taint analysis would report this as an attack.

Another run….

Example

```
→i = get_input();
  two = 2;
  if(i %2 == 0){
     j = i+two;
     l = k;
  } else {
     k = two*two;
     l = k;
  }
  jmp l;
```

Variable	Value (Int)	Taint Status (T/F)
i	7	T

Example

```
i = get_input();
two = 2;
if(i %2 == 0){
    j = i+two;
    l = k;
} else {
    k = two*two;
    l = k;
}
jmp l;
```

Variable	Value (Int)	Taint Status (T/F)
i	7	T
two	2	F

Example

```
i = get_input();
two = 2;
if(i %2 == 0){
    j = i+two;
    l = k;
} else {
    k = two*two;
    l = k;
}
jmp l;
```

Variable	Value (Int)	Taint Status (T/F)
i	7	T
two	2	F

Example

```
i = get_input();
two = 2;
if(i %2 == 0){
    j = i+two;
    l = k;
} else {
    k = two*two;
    l = k;
}
jmp l;
```

Variable	Value (Int)	Taint Status (T/F)
i	7	T
two	2	F
k	4	F

$$\frac{t_1 = \text{taint of } x_1 \quad t_2 = \text{taint of } x_2 \quad t = t_1 \vee t_2}{x_1 \square x_2 \Downarrow t}\ \text{OP}$$

Example

```
i = get_input();
two = 2;
if(i %2 == 0){
    j = i+two;
    l = k;
} else {
    k = two*two;
    l = k;
}
jmp l;
```

Variable	Value (Int)	Taint Status (T/F)
i	7	T
two	2	F
k	4	F
l	4	F

Example

```
i = get_input();
two = 2;
if(i %2 == 0){
    j = i+two;
    l = k;
} else {
    k = two*two;
    l = k;
}
jmp l;
```

Variable	Value (Int)	Taint Status (T/F)
i	7	T
two	2	F
k	4	F
l	4	F

$$\frac{\text{TAINT STATUS OF X IS } f}{\text{jmp } x \text{ OK}}\ \text{JMP}$$

Dynamic taint analysis says this is OK.

Current Trends

- Dynamic taint analysis on binary code
- Use emulator (e.g., TEMU) to inspect each instruction
  - What goes wrong if you don't look at each instruction?
- Most pressing issue: high overhead
  - CPU bound terrible, e.g., 30x for gzip
  - IO bound not so bad.

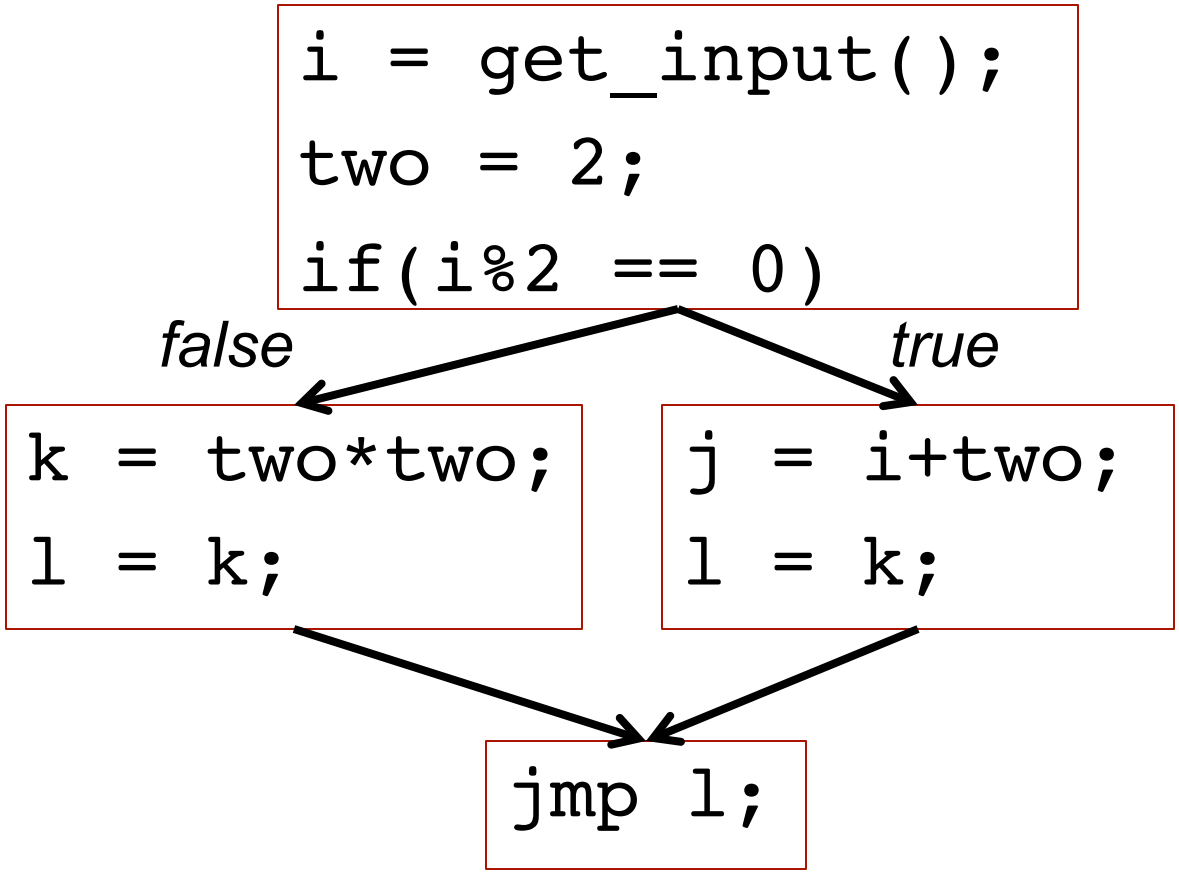
Static Taint Analysis

Analysis performed over *multiple paths* of a program

* Typically performed on a control flow graph (CFG): statements are nodes, and there is an edge between nodes if there is a possible transfer of control.

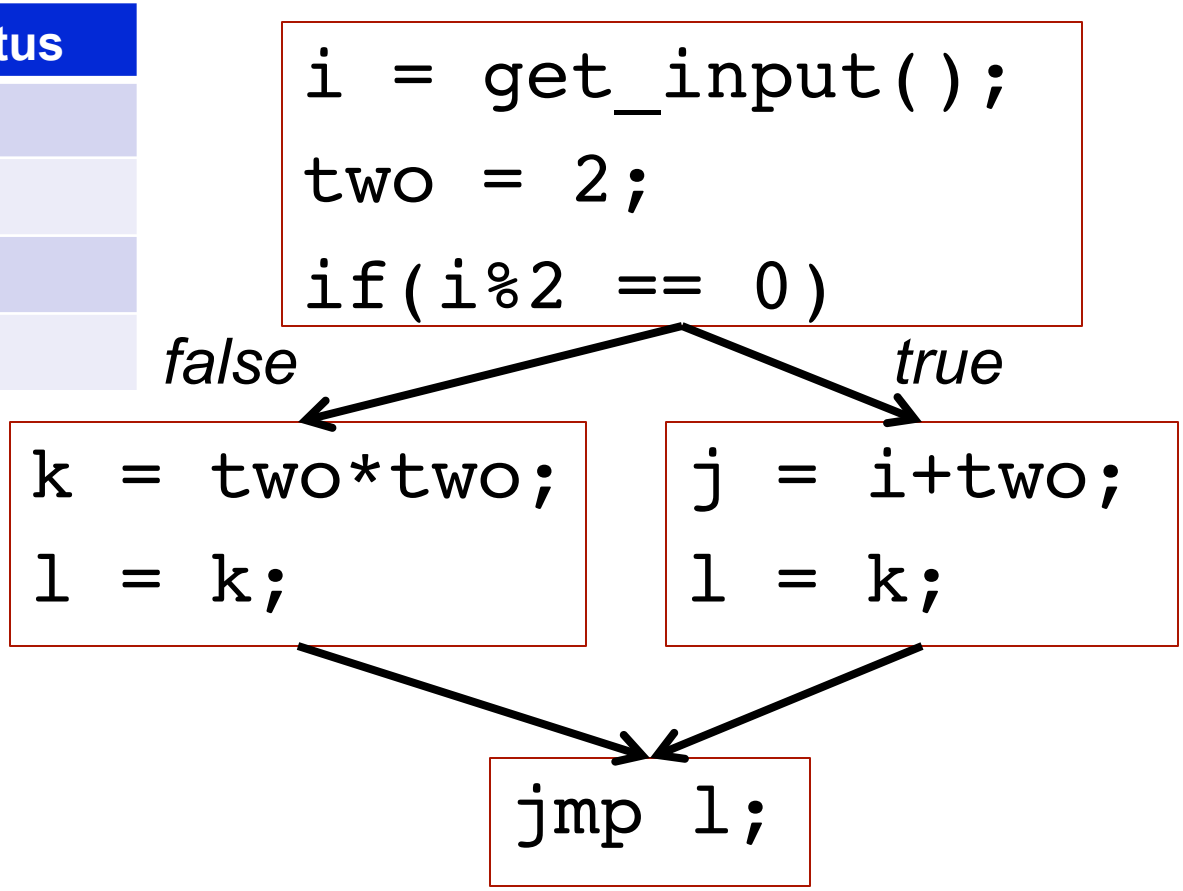
CFG

```
i = get_input();
two = 2;
if(i %2 == 0){
    j = i+two;
    l = k;
} else {
    k = two*two;
    l = k;
}
jmp l;
```



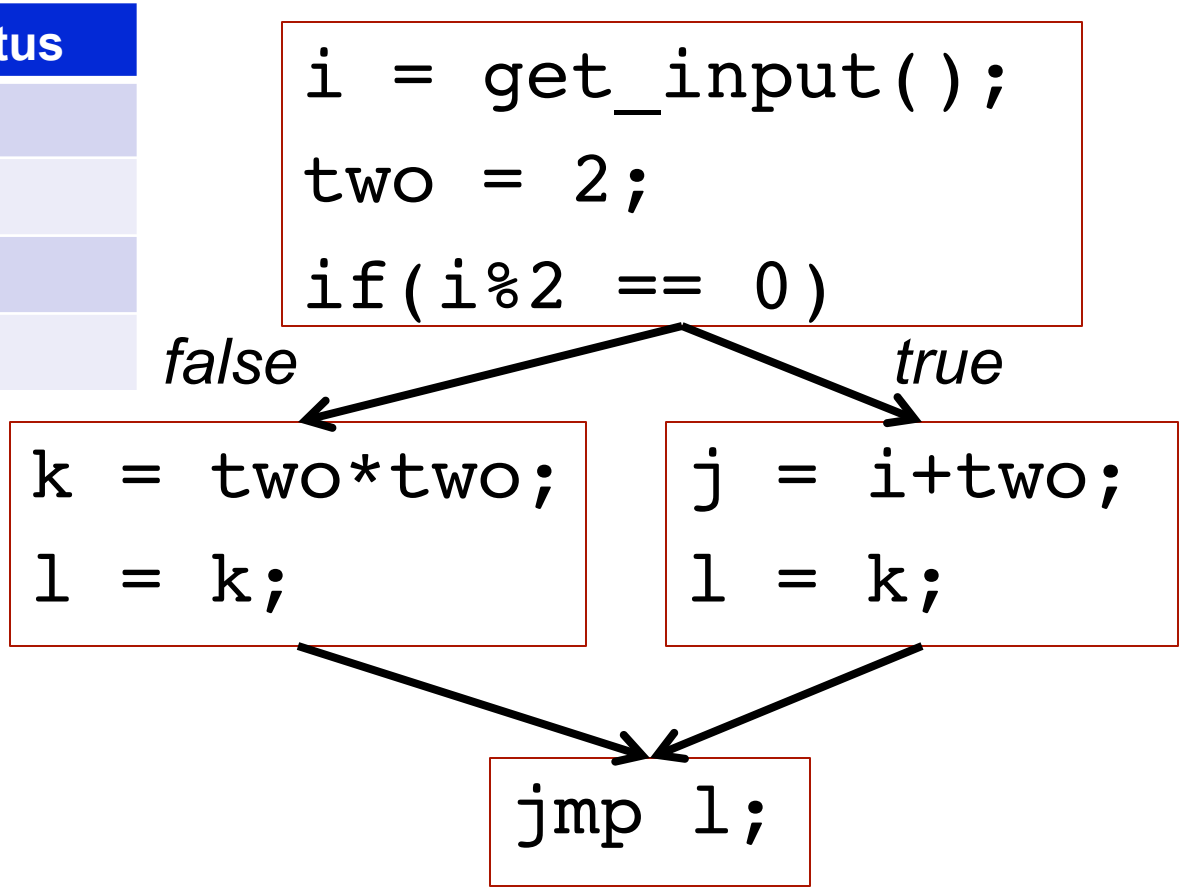
CFG

Variable	Taint Status
i	⊥
two	⊥
k	⊥
l	⊥

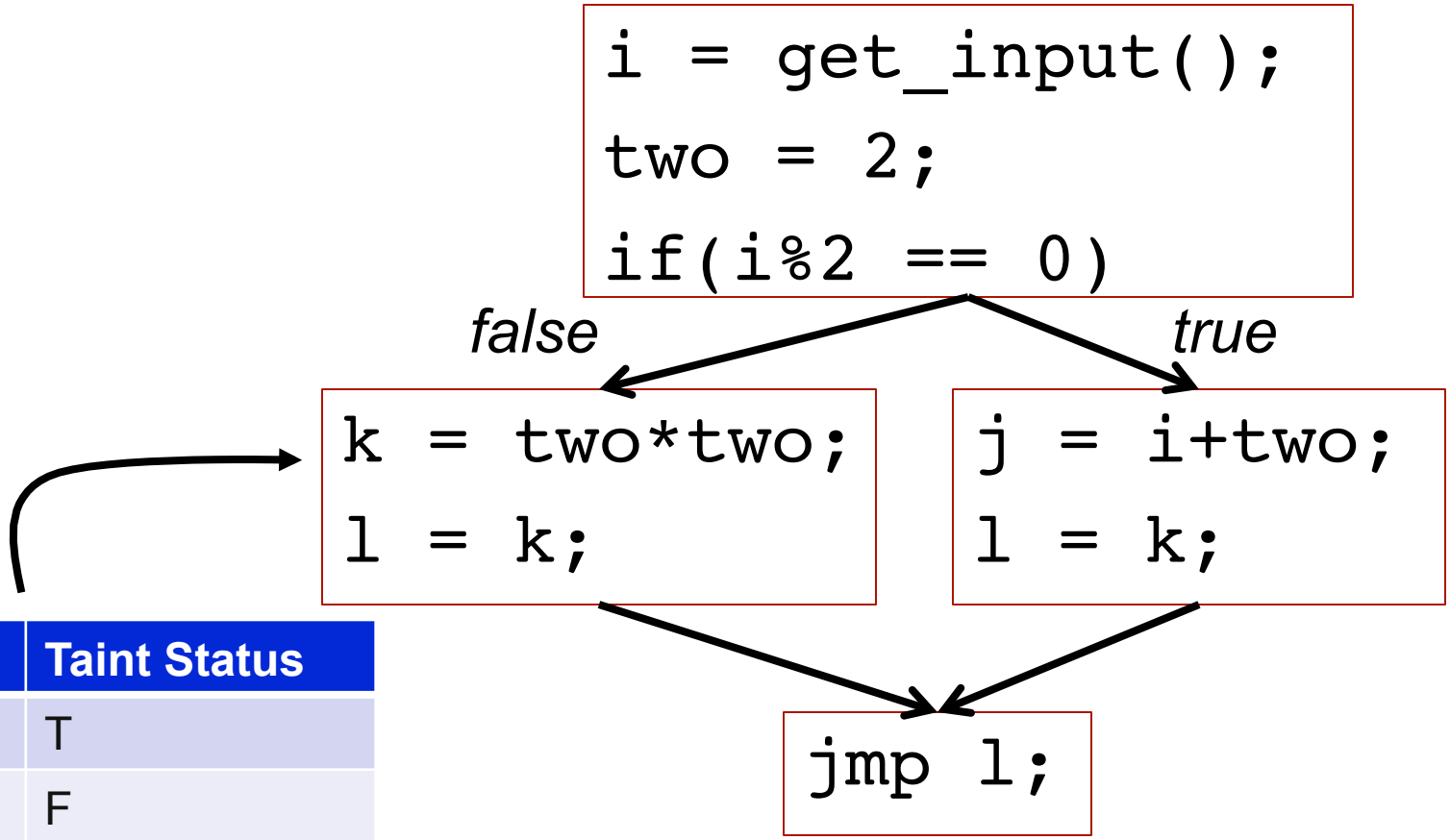


Walk Graph to Determine Taint

Variable	Taint Status
i	T
two	F
k	⊥
l	⊥

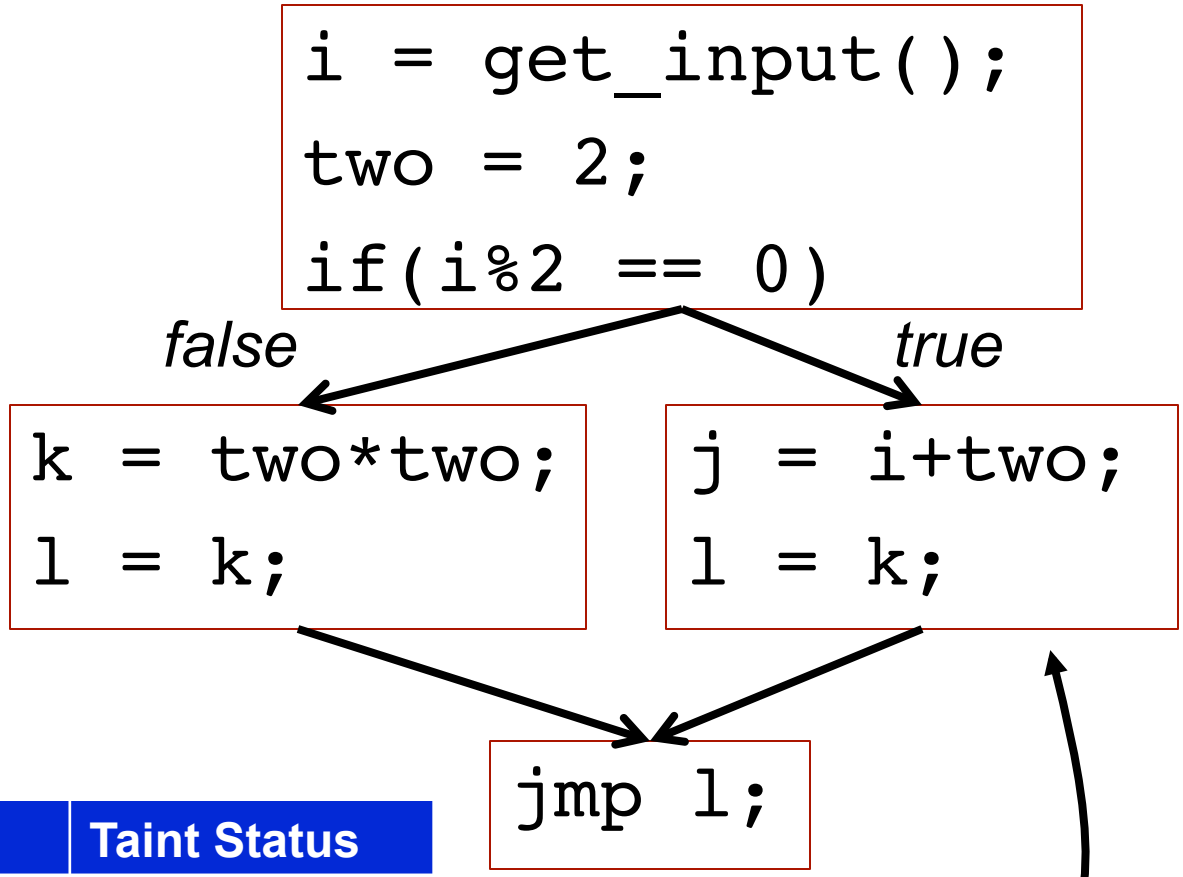


Walk Graph to Determine Taint



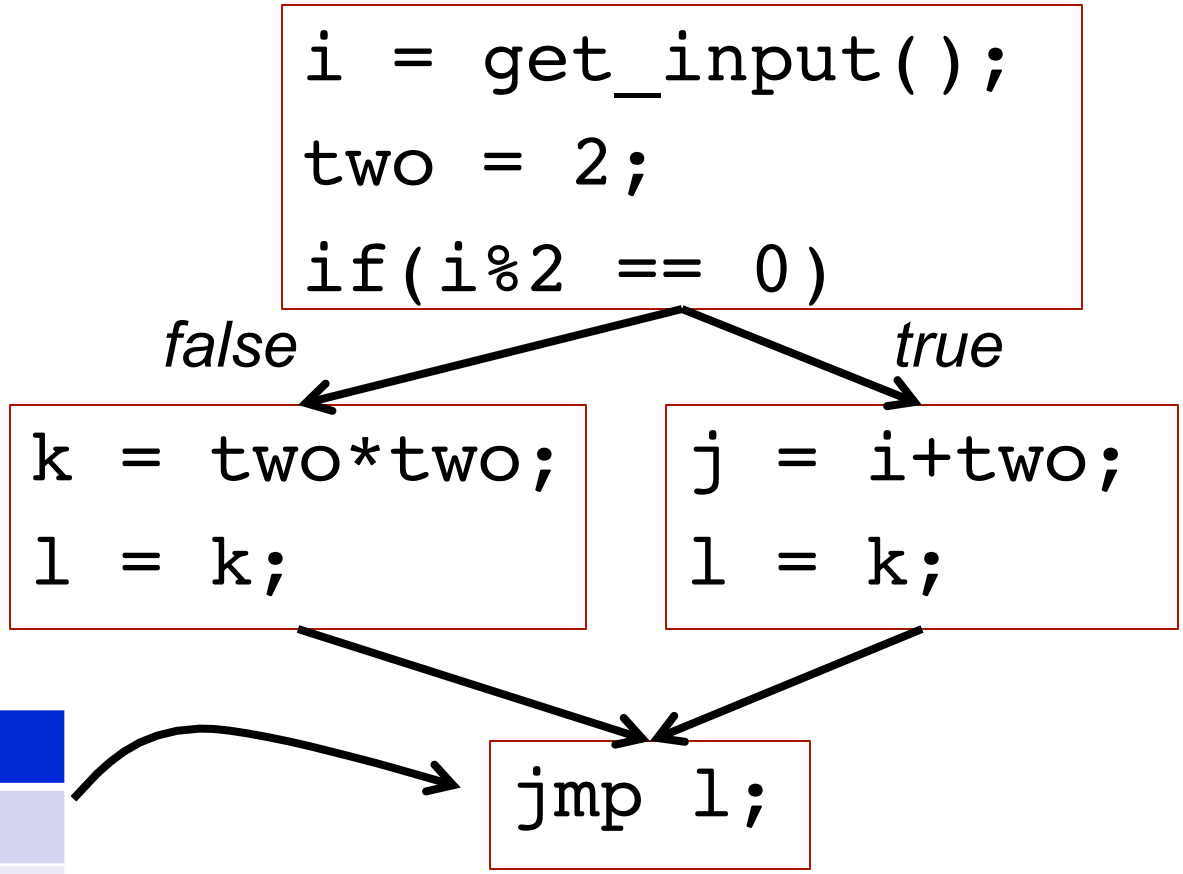
Variable	Taint Status
i	T
two	F
k	F
l	F

Walk Graph to Determine Taint



Variable	Taint Status
i	T
two	F
j	T
l	T

Walk Graph to Determine Taint



Variable	Taint Status
i	T
two	F
k	F
j	T
l	T

Confluence of paths:
We take most conservative value of "l"

Comparison

Dynamic

- Looks at a single path
- Determines exact taint values for run
- Must be run on each execution to detect attacks
- Combining multiple runs makes dynamic = static

Static

- Looks at multiple paths
- Must either over or under-approximate taint at confluence of paths
- Can be used to add monitoring code for only vulnerable paths

Some Limitations and Issues

- Control flow…

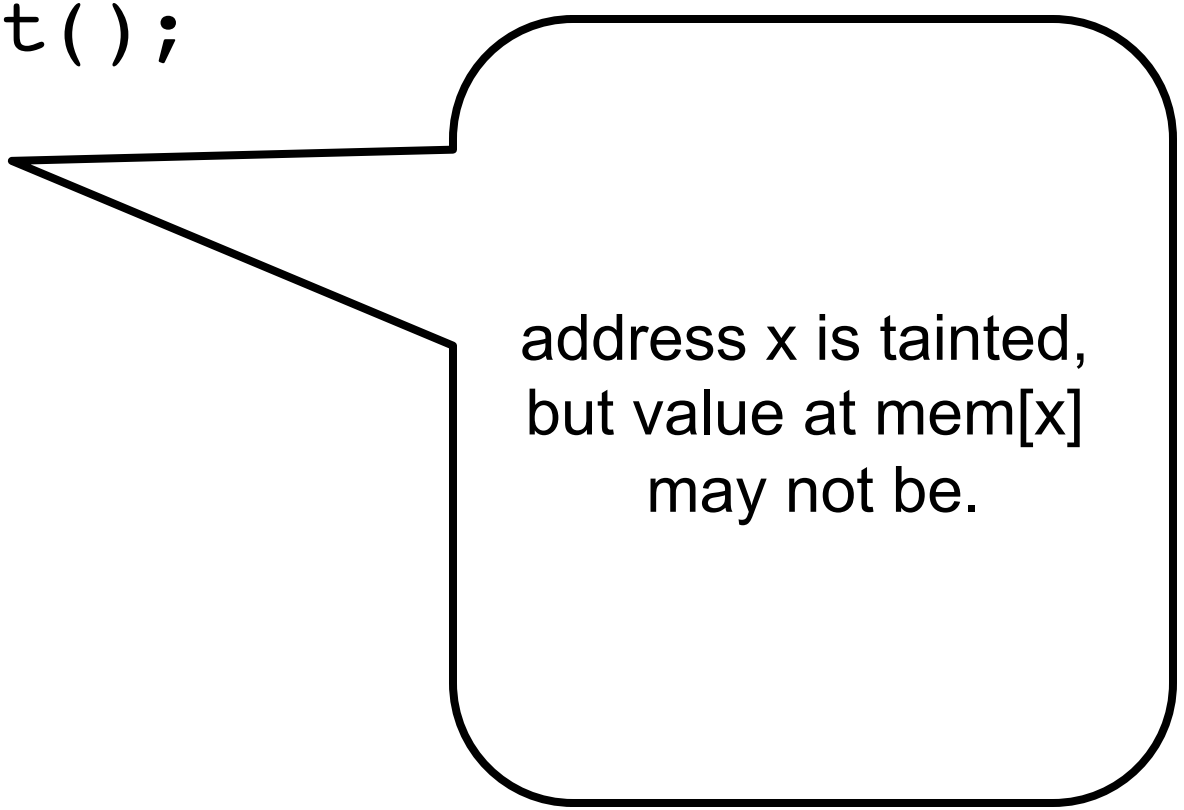
```
x = get_input();
if(x == 0)
   y = 0;
if(x == 1)
   y = 1;
if(x == 2)
   y = 2;
```

Value of y determined by tainted value x. Yet taint analysis would say y is *untainted* since it is *constant*.

Fixing requires control flow analysis, which requires static analysis. So dynamic-only approach cannot be "fixed" to solve problem.

Memory addresses vs. values: What should you do?

```
x = get_input();
a = load(x);
goto a;
```



Memory addresses vs. values: What should you do?

```
x = get_input();
a = load(x);
goto a;
```

Good or bad? Conservative says bad. But this breaks good programs, e.g., tcpdump uses a packet header value to index a function pointer table for the appropriate printer.

For more information

- "All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution (but might have been afraid to ask)"
  - By Ed Schwartz, Thanassis Averginos, David Brumley. At IEEE Security and Privacy Symposium, 2010

- "Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software"
  - By James Newsome and Dawn Song. At NDSS, 2005.

Taint Analysis Summary

- Data flow analysis to determine if value derived from user input
- “Any value derived from tainted data is data”
- “Users should not be able to determine jump target addresses”
- Some issues
  - Overhead
  - Tainted address vs. value
  - Control flow “taint”

That is all on taint analysis.

35