



Taint Analysis

Yajin Zhou (<http://yajin.org>)

Zhejiang University



Introduction



Secure Computing Systems

- Overall goal: Secure the data manipulated by a computing system
- Enforce a **security policy**
 - **Confidentiality**: Secret data does not leak to non-secret places
 - **Integrity**: High-integrity data is not influenced by low-integrity data



Information Flow

- **Goal of information flow analysis:**
Check whether information from one
"place" **propagates** to another "place"
 - For program analysis, "place" means, e.g.,
code location or **variable**
- **Complements techniques that impose limits on releasing information**
 - Access control lists
 - Cryptography



Example: Confidentiality

Credit card number should not leak to visible

```
var creditCardNb = 1234;  
var x = creditCardNb;  
var visible = false;  
if (x > 1000) {  
    visible = true;  
}
```

Example: Confidentiality



Credit card number should not leak to visible

```
var creditCardNb = 1234;  
var x = creditCardNb;  
var visible = false;  
if (x > 1000) {  
    visible = true;  
}
```

Secret information
propagates to x

Secret information
(partly) propagates
to visible

Example: Integrity



userInput should not influence who becomes president

```
var designatedPresident = "Michael";  
var x = userInput();  
var designatedPresident = x;
```

Example: Integrity



userInput should not influence who becomes president

```
var designatedPresident = "Michael";  
var x = userInput();  
var designatedPresident = x;
```

↑
Low-integrity information
propagates to high-integrity
variable

Example: Integrity



userInput should not influence who becomes president

```
var designatedPresident = "Michael";  
var x = userInput();  
if (x.length === 5) {  
    var designatedPresident = "Paul";  
}
```

Example: Integrity



userInput should not influence who becomes president

```
var designatedPresident = "Michael";  
var x = userInput();  
if (x.length === 5) {  
    var designatedPresident = "Paul";  
}
```

Low-integrity information
propagates to high-integrity
variable

Confidentiality vs. Integrity



Confidentiality and integrity are dual problems for information flow analysis

(Focus of this lecture: Confidentiality)

Tracking Security Labels



How to analyze the flow of information?

- Assign to each value some **meta information** that tracks the **secrecy** of the value
- **Propagate meta information** on program operations

Non-Interference



Property that information flow analysis aims to ensure:

Confidential data does not interfere with public data

- Variation of confidential input **does not cause** a variation of public output
- Attacker **cannot observe any difference** between two executions that differ only in their confidential input



Information Flow Policy

Flow Relation



- Partial order on security classes defines a **flow relation**
- Program is **secure** if and only if **all information flows** are described by the **flow relation**
- Intuition: **No flow from higher to lower security class**

Information Flow Policy



Policy specifies **secrecy of values** and which **flows** are allowed:

- Lattice of security classes
- **Sources** of secret information
- Untrusted **sinks**

Goal:

**No flow from
source to sink**

Information Flow Policy



Policy specifies **secrecy of values** and which **flows** are allowed:

- Lattice of security classes
- **Sources** of secret information
- Untrusted **sinks**

Goal:
No flow from
source to sink

```
var creditCardNb = 1234;  
var x = creditCardNb;  
var visible = false;  
if (x > 1000) {  
    visible = true;  
}
```

Information Flow Policy

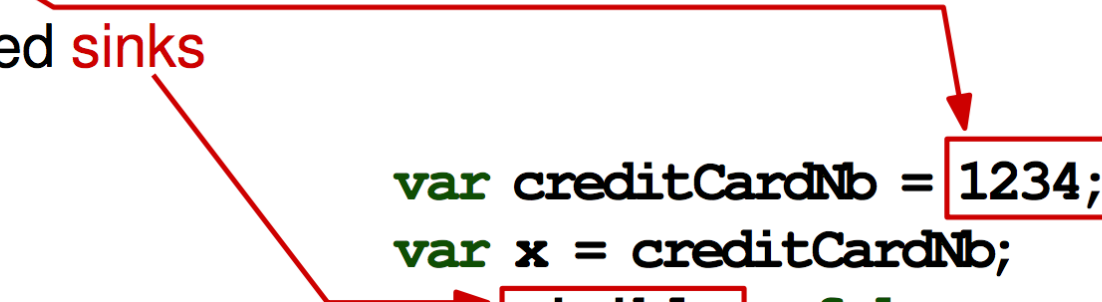


Policy specifies **secrecy of values** and which **flows** are allowed:

- Lattice of security classes
- **Sources** of secret information
- Untrusted **sinks**

Goal:
No flow from
source to sink

```
var creditCardNb = 1234;  
var x = creditCardNb;  
var visible = false;  
if (x > 1000) {  
    visible = true;  
}
```



The diagram illustrates the flow of secret information from a source to a sink. A red arrow originates from the word 'Sources' in the list above and points to the value '1234' in the first line of code, which is enclosed in a red box. Another red arrow originates from the word 'sinks' in the list and points to the variable 'visible' in the third line of code, which is also enclosed in a red box. This visualizes the flow of secret information from the source (credit card number) to the sink (visibility flag).

Declassification



- "No flow from high to low" is **impractical**
- E.g., code that checks password against a hash value propagates information to subsequence statements

But: This is intended

```
var password = .. // secret
if (hash(password) == 23) {
    // continue normal program execution
} else {
    // display message: incorrect password
}
```

Declassification



- "No flow from high to low" is **impractical**
- E.g., code that checks password against a hash value propagates information to subsequence statements

But: This is intended

```
var password = .. // secret
if (hash(password) == 23) {
    // continue normal program execution
} else {
    // display message: incorrect password
}
```

Declassification: Mechanism to remove or lower security class of a value



Analyze Information Flows

Analyzing Information Flows



Given an information flow policy,
analysis **checks for policy violations**

Applications:

- Detect **vulnerable code** (e.g, potential SQL injections)
- Detect **malicious code** (e.g., privacy violations)
- Check if program **behaves as expected** (e.g., secret data should never be written to console)

Explicit vs. Implicit Flows



- **Explicit flows:** Caused by data flow dependence
- **Implicit flows:** Caused by control flow dependence

Explicit vs. Implicit Flows



- **Explicit flows:** Caused by data flow dependence
- **Implicit flows:** Caused by control flow dependence

```
var creditCardNb = 1234;  
var x = creditCardNb;  
var visible = false;  
if (x > 1000) {  
    visible = true;  
}
```


Explicit vs. Implicit Flows



- **Explicit flows:** Caused by data flow dependence
- **Implicit flows:** Caused by control flow dependence

```
var creditCardNb = 1234;  
var x = creditCardNb;  
var visible = false;  
if (x > 1000) {  
    visible = true;  
}
```

Explicit flow from
creditCardNb to x

Implicit flow from
x > 1000 to visible

Static and Dynamic Analysis



■ Static information flow analysis

- **Overapproximate** all possible data and control flow dependences
- Result: Whether information "**may flow**" from secret source to untrusted sink

■ Dynamic information flow analysis

- Associate security labels ("**taint markings**") with **memory locations**
- **Propagate** labels at **runtime**

Taint Sources and Sinks



■ Possible sources:

- Variables
- Return values of a particular function
- Data from a type of I/O stream
- Data from a particular I/O stream

Taint Sources and Sinks



■ Possible sources:

- Variables
- Return values of a particular function
- Data from a type of I/O stream
- Data from a particular I/O stream

■ Possible sinks:

- Variables
- Parameters given to a particular function
- Instructions of a particular type (e.g., jump instructions)

Taint Sources and Sinks



■ Possible sources:

- Variables
- Return values of a particular function
- Data from a type of I/O stream
- Data from a particular I/O stream

■ Possible sinks:

- Variables
- Parameters given to a particular function
- Instructions of a particular type (e.g., jump instructions)

Report illegal flow if taint marking flows to a sink where it should not flow

Taint Propagation



1) Explicit flows

For every operation that produces a new value, **propagate labels of inputs to label of output:**

$$label(result) \leftarrow label(inp_1) \oplus \dots \oplus label(inp_2)$$