

# Retrofitting Legacy Code for Security

**Vinod Ganapathy**

Computer Sciences Department  
University of Wisconsin-Madison

[vg@cs.wisc.edu](mailto:vg@cs.wisc.edu)

# Principle of Design for Security

**To create a secure system, design it to be secure from the ground up**

- Historic example:
  - MULTICS [Corbato *et al.* '65]
- More recent examples:
  - Operating systems
  - Database servers

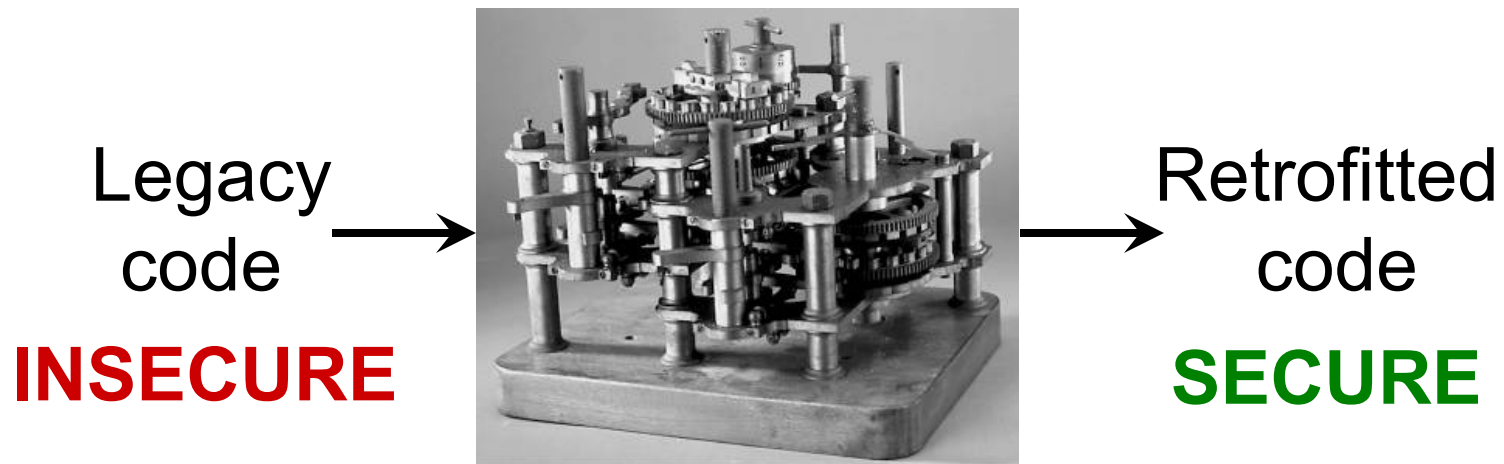
# Relevance of the Principle today

**Most deployed software is not designed for security**

- Deadline-driven software development
  - **Design.Build.(Patch)\*** is here to stay
- Diverse/Evolving security requirements
  - MULTICS security study [Karger and Schell, '72]

# Retrofitting legacy code

**Need systematic techniques to retrofit legacy code for security**

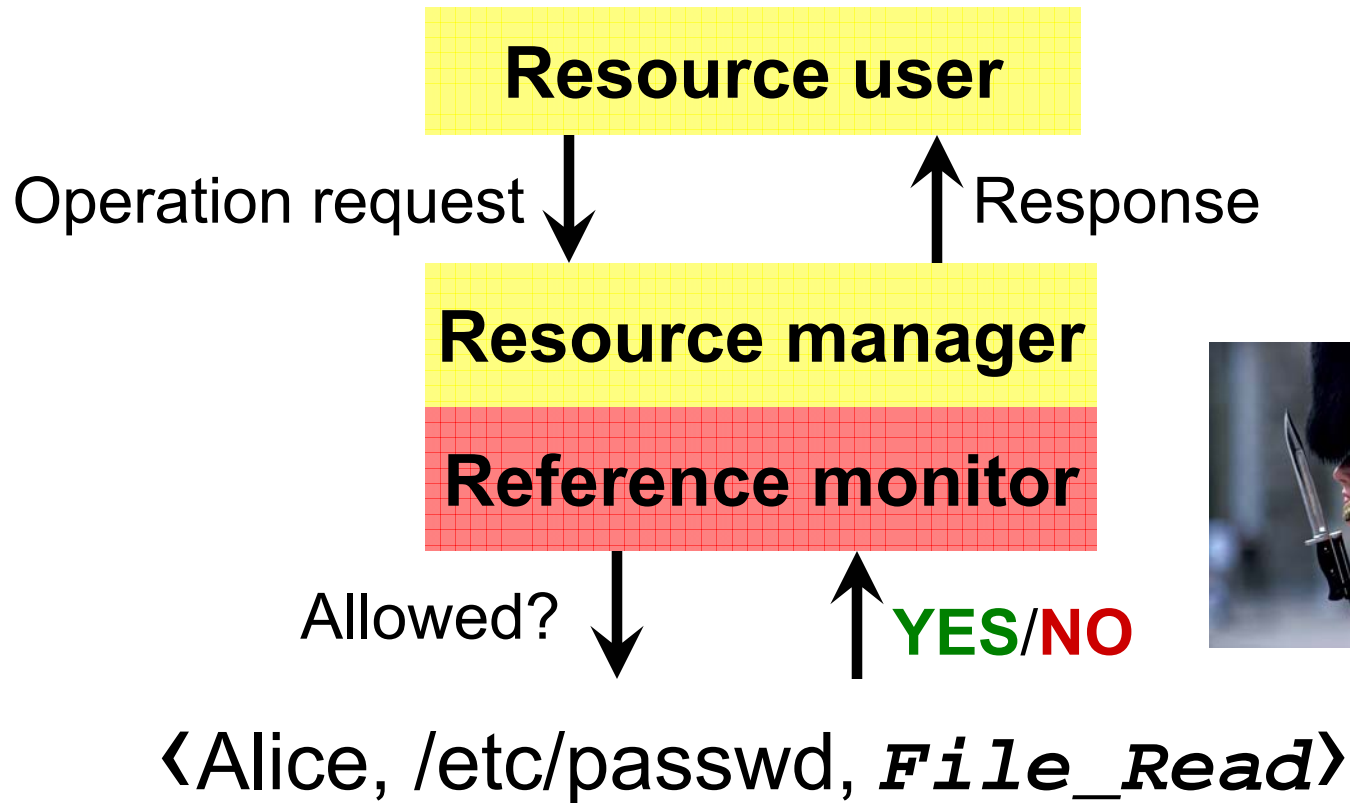


# Retrofitting legacy code

**Need systematic techniques to retrofit legacy code for security**

- Enforcing type safety
  - CCured [Necula *et al.* '02]
- Partitioning for privilege separation
  - PrivTrans [Brumley and Song, '04]
- **Enforcing authorization policies**

# Enforcing authorization policies



# Retrofitting for authorization

- Mandatory access control for Linux
  - Linux Security Modules [Wright *et al.*, '02]
  - SELinux [Loscocco and Smalley, '01]
- **Painstaking, manual procedure**
  - Trusted X, Compartmented-mode workstation, X11/SELinux [Epstein *et al.*, '90][Berger *et al.*, '90][Kilpatrick *et al.*, '03]
- Java Virtual Machine/SELinux [Fletcher, '06]
- IBM Websphere/SELinux [Hocking *et al.*, '06]

# Contributions

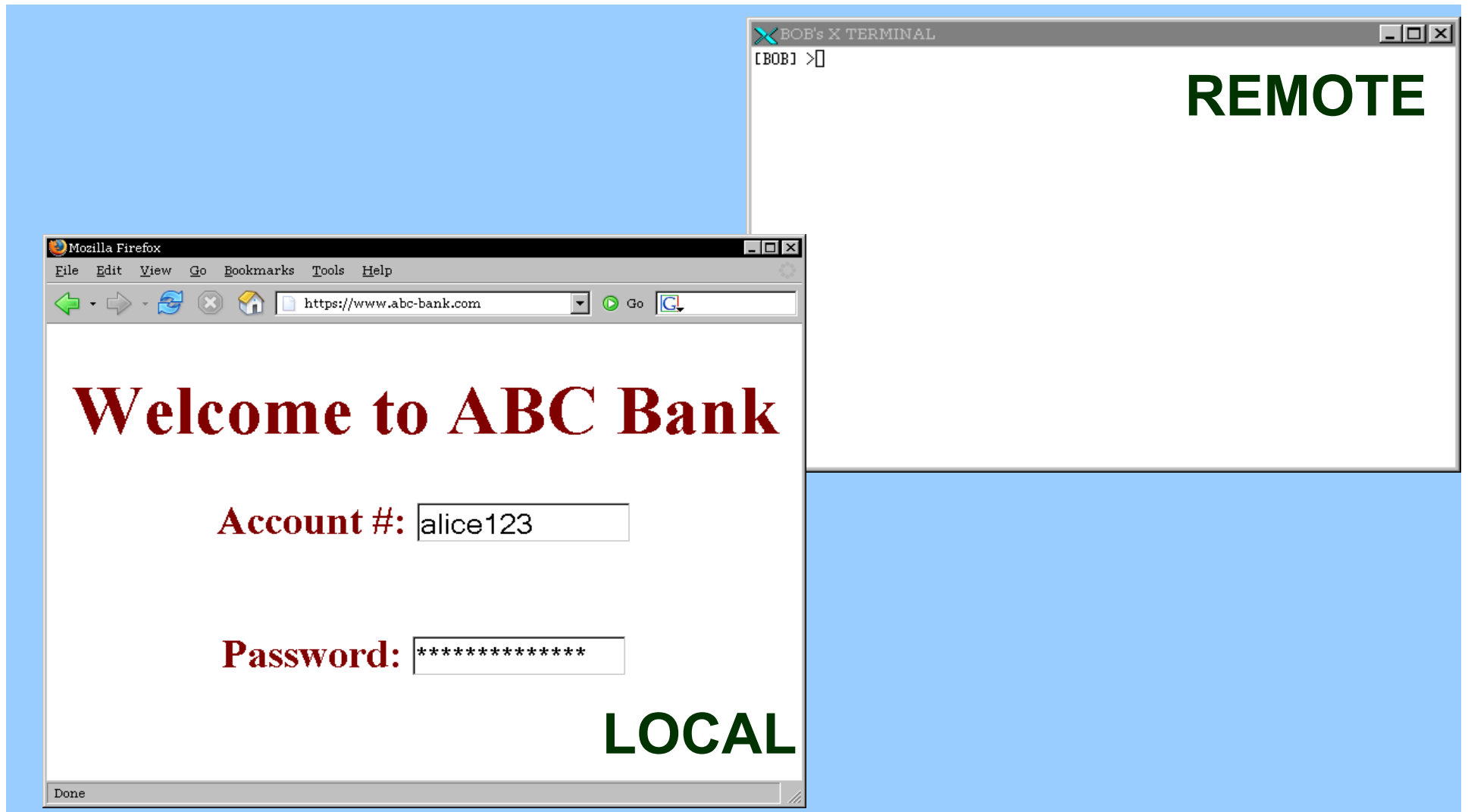
## Analyses and transformations for authorization policy enforcement

- **Fingerprints**: New abstraction to represent security-sensitive operations
- Reduced effort to retrofit legacy code for authorization policy enforcement
  - From **several years** to a **few hours**
  - Applied to X server, Linux kernel, PennMUSH

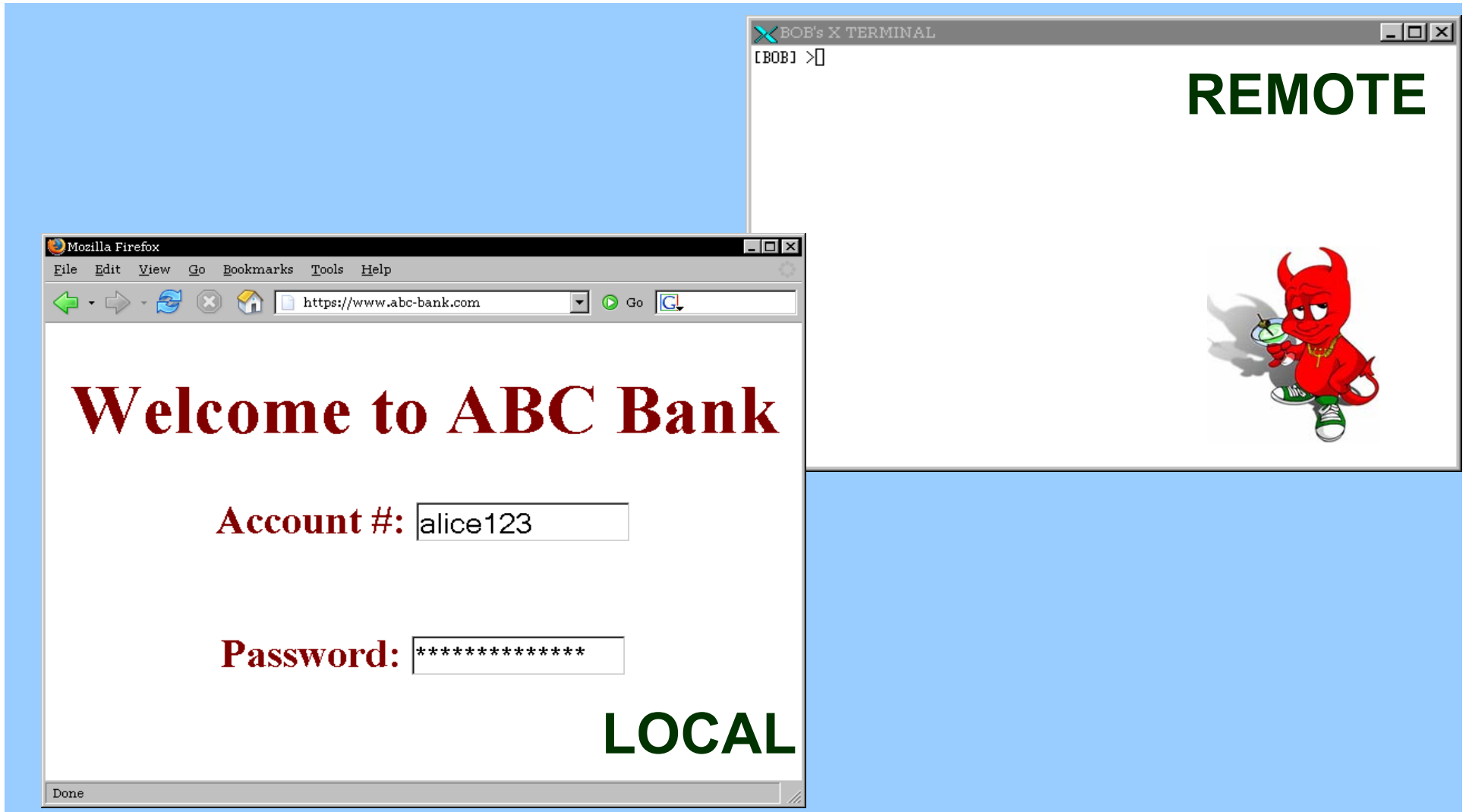
# Outline

- Motivation
- Problem
  - Example
  - Retrofitting legacy code: Lifecycle
- Solution

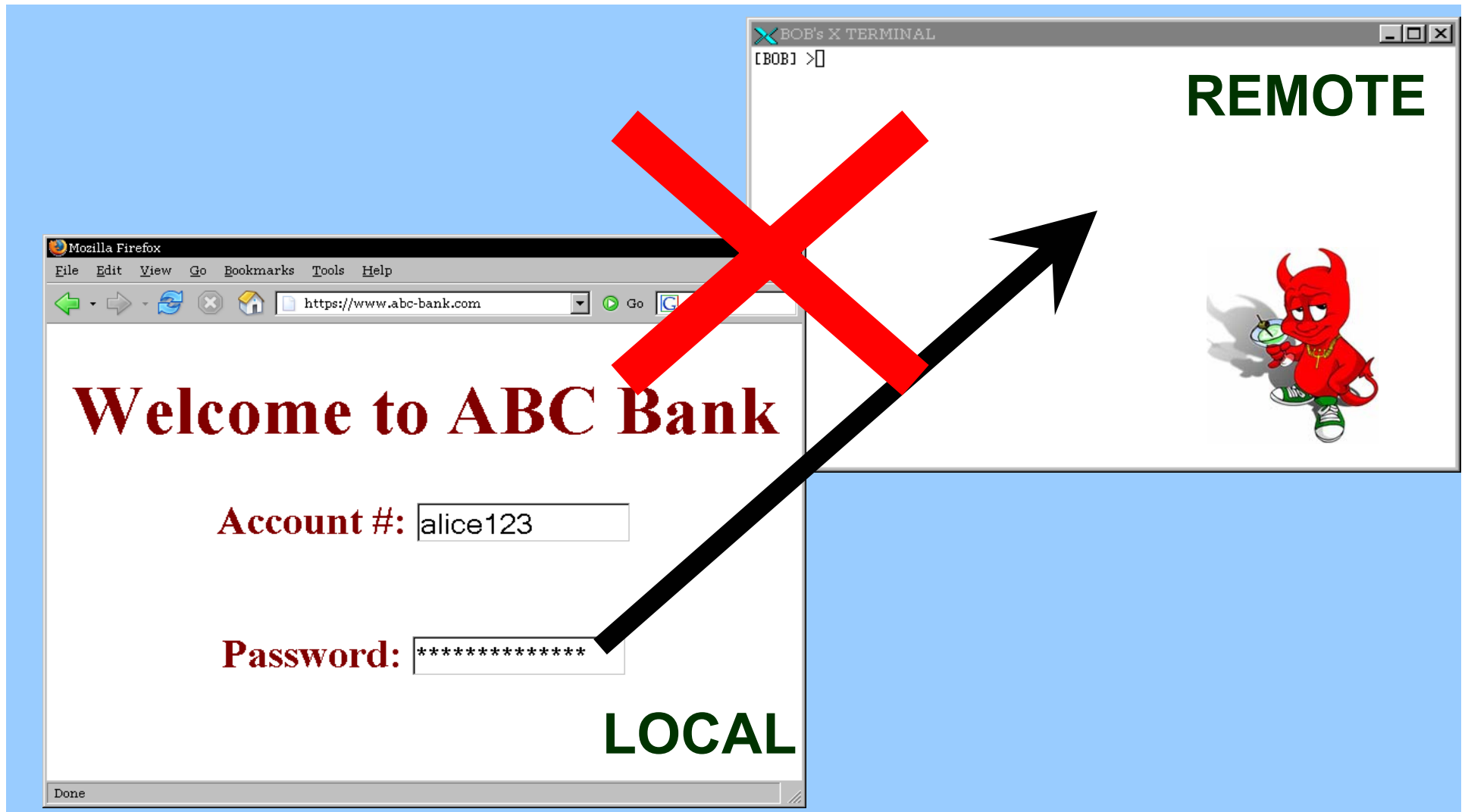
# X server with multiple X clients



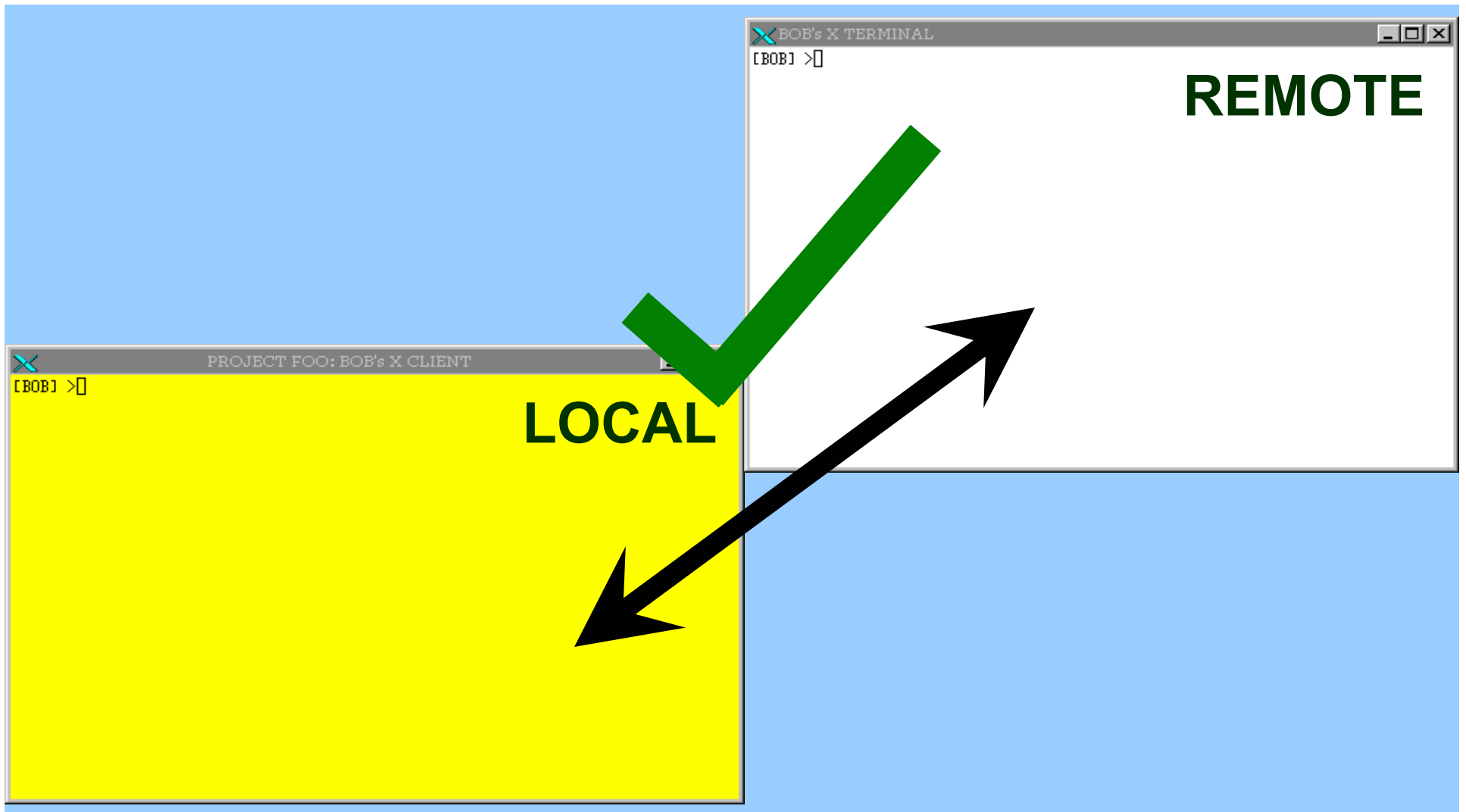
# Malicious remote X client



# Undesirable information flow



# Desirable information flow



# Other policies to enforce

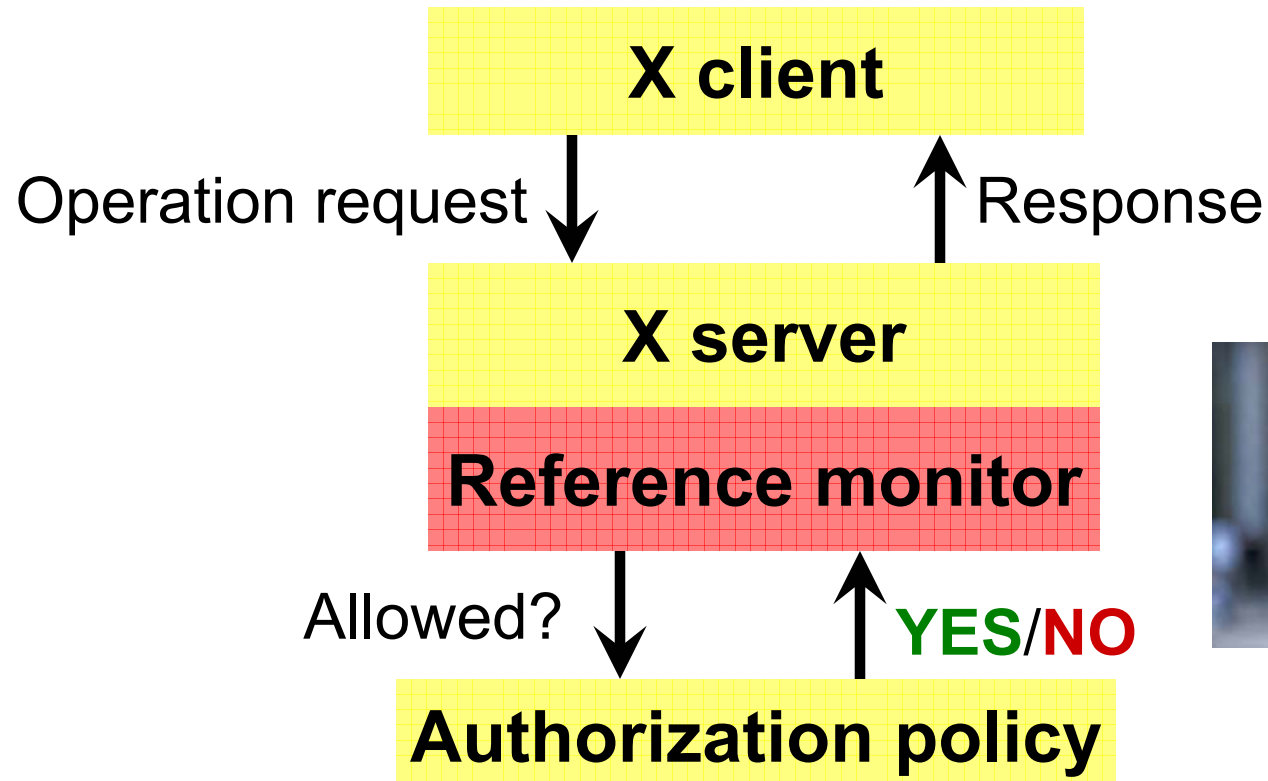
- Prevent unauthorized
  - Copy and paste
  - Modification of inputs meant for other clients
  - Changes to window settings of other clients
  - Retrieval of bitmaps: Screenshots

[Berger *et al.*, '90]

[Epstein *et al.*, '90]

[Kilpatrick *et al.*, '03]

# X server with authorization



# Outline

- Motivation
- Problem
  - Example
  - Retrofitting legacy code: Lifecycle
- Solution

# Retrofitting lifecycle



1. Identify security-sensitive operations
2. Locate where they are performed in code
3. Instrument these locations

## Security-sensitive operations

***Input\_Event***

*Create*

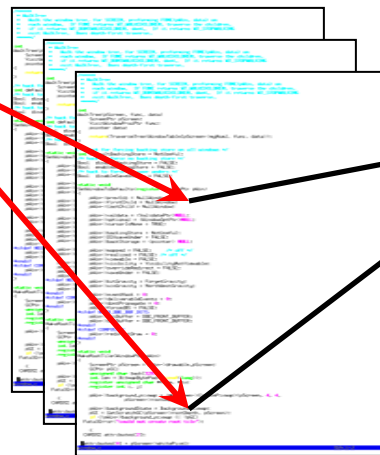
*Destroy*

*Copy*

*Paste*

*Map*

## Source Code



## Policy checks

Can the client  
receive this  
*Input\_Event*?

# Problems



## ■ Time-consuming

- X11/SELinux ~ 2 years [Kilpatrick *et al.*, '03]
- Linux Security Modules ~ 2 years [Wright *et al.*, '02]

## ■ Error-prone [Zhang *et al.*, '02][Jaeger *et al.*, '04]

- Violation of complete mediation
- Time-of-check to Time-of-use bugs

# Our approach



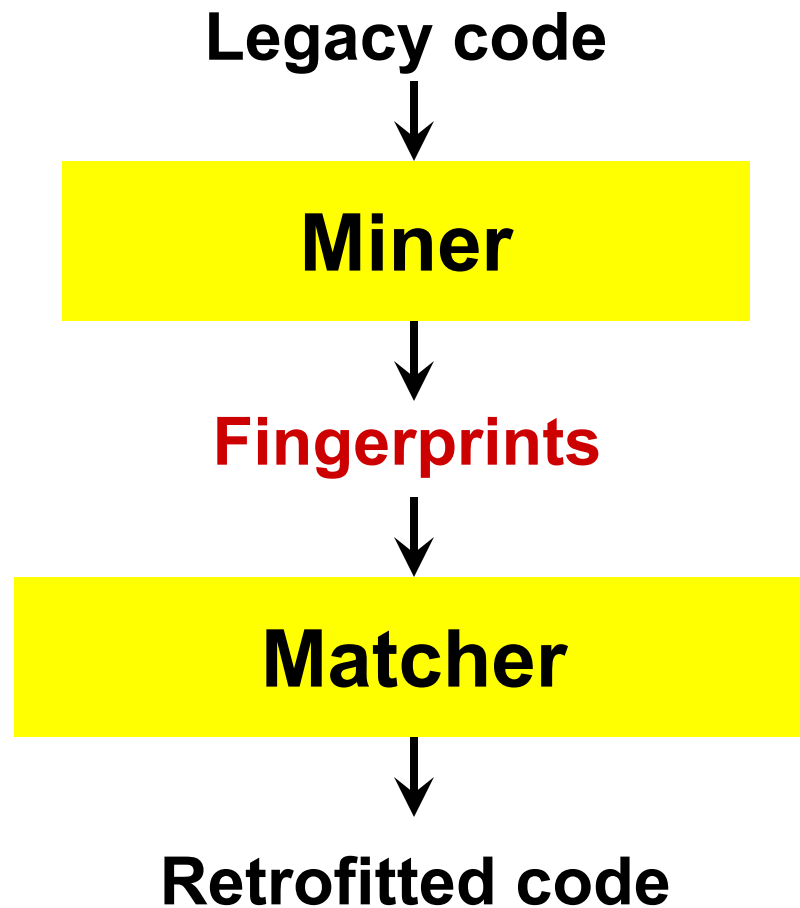
## Reduces manual effort

- Retrofitting takes just **a few hours**
  - Automatic analysis: ~ minutes
  - Interpreting results: ~ hours

## Reduces errors

- Basis to prove security of retrofitted code

# Approach overview



# Outline

- Motivation
- Problem
- **Solution**
  - **Fingerprints**
  - Dynamic fingerprint mining
  - Static fingerprint mining

**[CCS'05]**

# What are fingerprints?



## **Code-level signatures of security-sensitive operations**

- Resource accesses that are unique to a security-sensitive operation
- Denote key steps needed to perform the security-sensitive operation on a resource

# Examples of fingerprints

- *Input\_Event* :-

*Cmp* **xEvent** ->type == KeyPress

## Security-sensitive operations

***Input\_Event***

*Create*

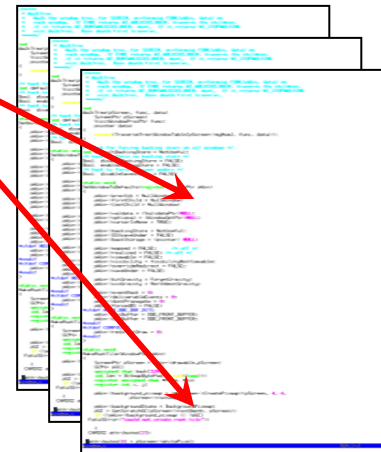
*Destroy*

*Copy*

*Paste*

*Map*

## Source Code



# Examples of fingerprints

- *Input\_Event* :-  
**Cmp** xEvent->type == KeyPress
- *Input\_Event* :-  
**Cmp** xEvent->type == MouseMove
- *Map* :-  
**Set** Window->mapped **to** True &  
**Set** xEvent->type **to** MapNotify
- *Enumerate* :-  
**Read** Window->firstChild &  
**Read** Window->nextSib &  
**Cmp** Window ≠ 0

# Fingerprint matching

```
Enumerate :- Read Window->firstChild &  
             Read Window->nextSib &  
             Cmp Window ≠ 0
```

```
MapSubWindows(Window *pParent, Client *pClient) {  
    Window *pWin;  
    ...  
    // Run through linked list of child windows  
    pWin = pParent->firstChild; ...  
    for (; pWin != 0; pWin=pWin->nextSib) {  
        ...  
        // Code that maps each child window  
        ...  
    }  
}
```

Performs *Enumerate*

# Placing authorization checks

- X server function **MapSubWindows**

```
MapSubWindows(Window *pParent, Client *pClient) {
    Window *pWin;
    ...
    // Run through linked list of child windows
    if CHECK(pClient, pParent, Enumerate) == ALLOWED {
        pWin = pParent->firstChild; ...
        for (; pWin != 0; pWin=pWin->nextSib) {
            ...
            // Code that maps each child window
            ...
        }
    } else { HANDLE_FAILURE }
}
```

# Fingerprint matching

- Currently employ simple pattern matching
- More sophisticated matching possible
  - Metacompilation [Engler *et al.*, '01]
  - MOPS [Chen and Wagner, '02]
- Inserting authorization checks is akin to static aspect-weaving [Kiczales *et al.*, '97]
- Other aspect-weaving techniques possible
  - Runtime aspect-weaving

# Outline

- Motivation
- Problem
- **Solution**
  - Fingerprints
  - **Dynamic fingerprint mining**
  - Static fingerprint mining

**[Oakland'06]**

# Dynamic fingerprint mining

## Security-sensitive operations

*Input\_Event*

*Create*

*Destroy*

*Copy*

*Paste*

*Map*

## Source Code



## Output: Fingerprints

*Input\_Event* :-

**Cmp** `xEvent->type == KeyPress`

# Dynamic fingerprint mining

- **Security-sensitive operations** [NSA'03]

<i>Input_Event</i>	Input to window from device
<i>Create</i>	Create new window
<i>Destroy</i>	Destroy existing window
<i>Map</i>	Map window to console

- Use this information to induce the program to perform security-sensitive operations

# Problem definition

- **S**: Set of security-sensitive operations
- **D**: Descriptions of operations in **S**
- **R**: Set of resource accesses
  - *Read/Set/Cmp* of `Window/xEvent`
- Each  $s \in \mathbf{S}$  has a fingerprint
  - A fingerprint is a subset of **R**
  - Contains a resource access unique to **s**
- **Problem**: Find fingerprints for each security-sensitive operation in **S** using **D**

# Traces contain fingerprints



## Security-sensitive operations

***Input\_Event***

*Create*

*Destroy*

*Copy*

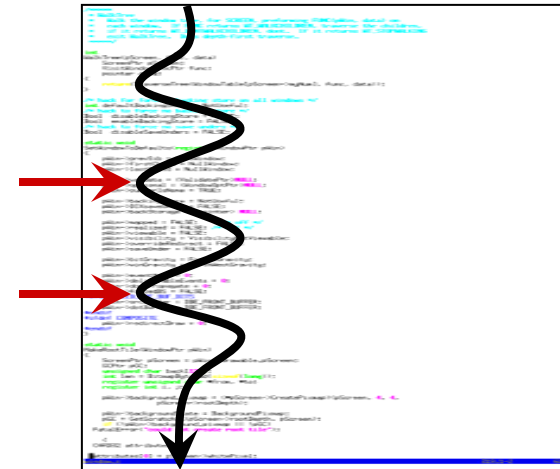
*Paste*

*Map*

## Source Code



## Runtime trace



- Induce security-sensitive operation
  - Typing to window will induce *Input\_Event*
- Fingerprint **must** be in runtime trace
  - ***Cmp*** `xEvent->type == KeyPress`

# Compare traces to localize



Security-sensitive operations

***Input\_Event***

*Create*

*Destroy*

*Copy*

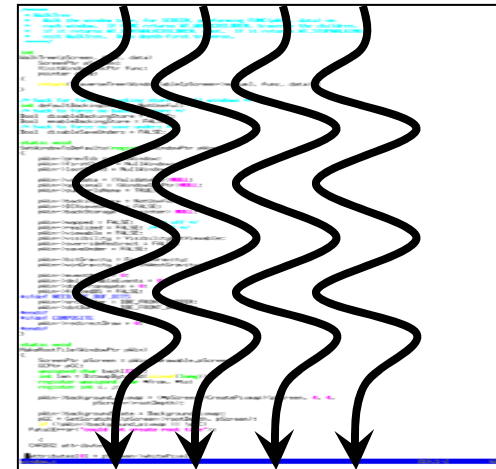
*Paste*

*Map*

Source Code



Runtime trace



- Localize fingerprint in trace
  - Trace difference and intersection

# Runtime traces

- Trace the program and record reads/writes to resource data structures
  - **Window** and **xEvent** in our experiments
- Example: from X server startup  
(In function **SetWindowtoDefaults**)
  - Set Window->prevSib to 0**
  - Set Window->firstChild to 0**
  - Set Window->lastChild to 0**
  - ...about 1400 such resource accesses

# Using traces for fingerprinting

- Obtain traces for each security-sensitive operation
  - Series of controlled tracing experiments
- Examples
  - Typing to keyboard generates *Input\_Event*
  - Creating new window generates *Create*
  - Creating window also generates *Map*
  - Closing existing window generates *Destroy*

# Comparison with “diff” and “∩”

Annotation is a manual step

	Open xterm	Close xterm	Move xterm	Open browser	Switch windows
<i>Create</i>	✓			✓	
<i>Destroy</i>		✓		✓	
<i>Map</i>	✓		✓	✓	
<i>Unmap</i>		✓		✓	
<i>Input_Event</i>			✓		✓

# Comparison with “diff” and “ $\cap$ ”

Perform same set operations on resource accesses

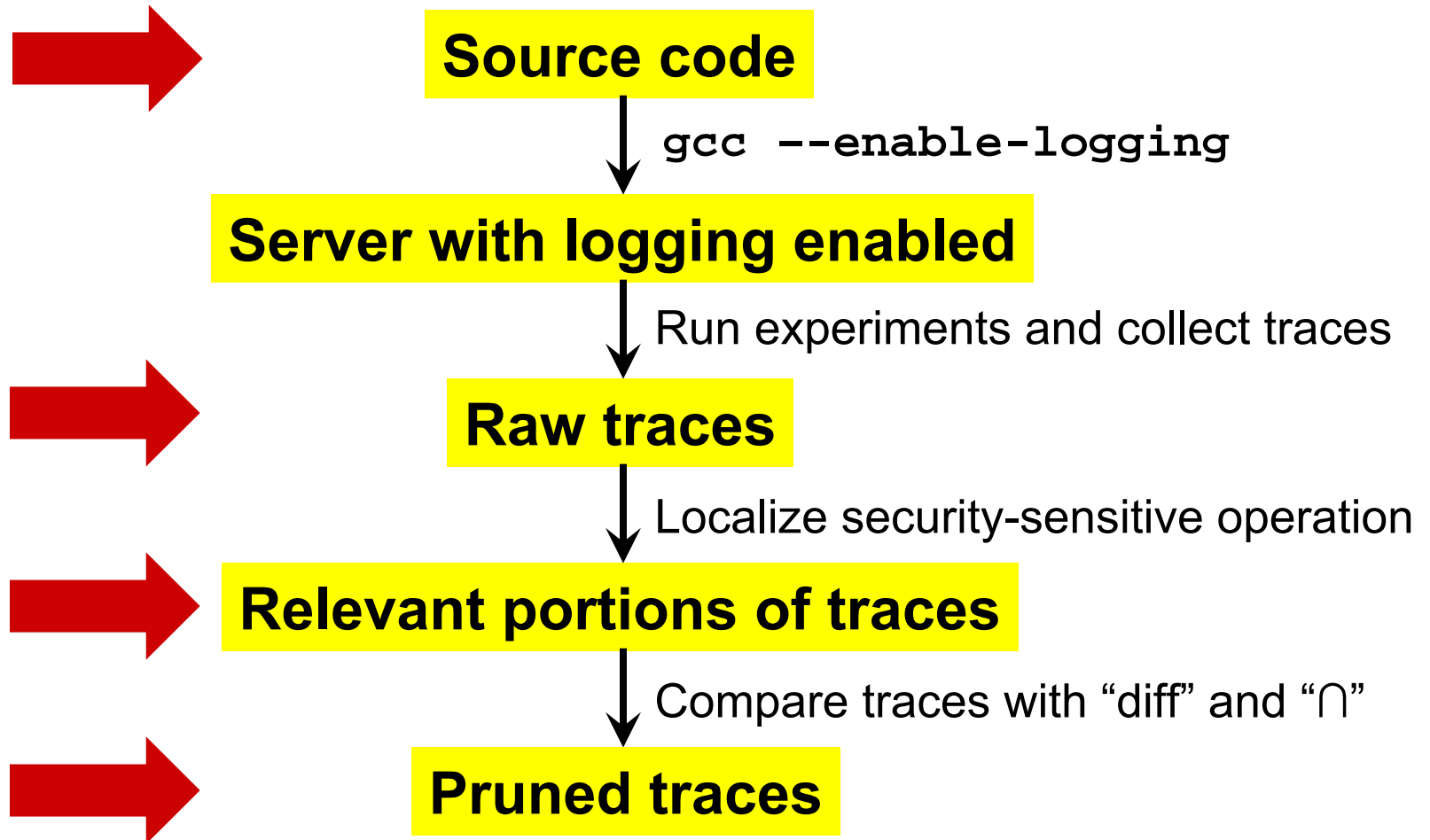
	Open xterm	Close xterm	Move xterm	Open browser	Switch windows
<i>Create</i>	✓			✓	
<i>Destroy</i>		✓		✓	
<i>Map</i>	✓		✓	✓	
<i>Unmap</i>		✓		✓	
<i>Input_Event</i>			✓		✓

*Create* = Open xterm  $\cap$  Open browser - Move xterm

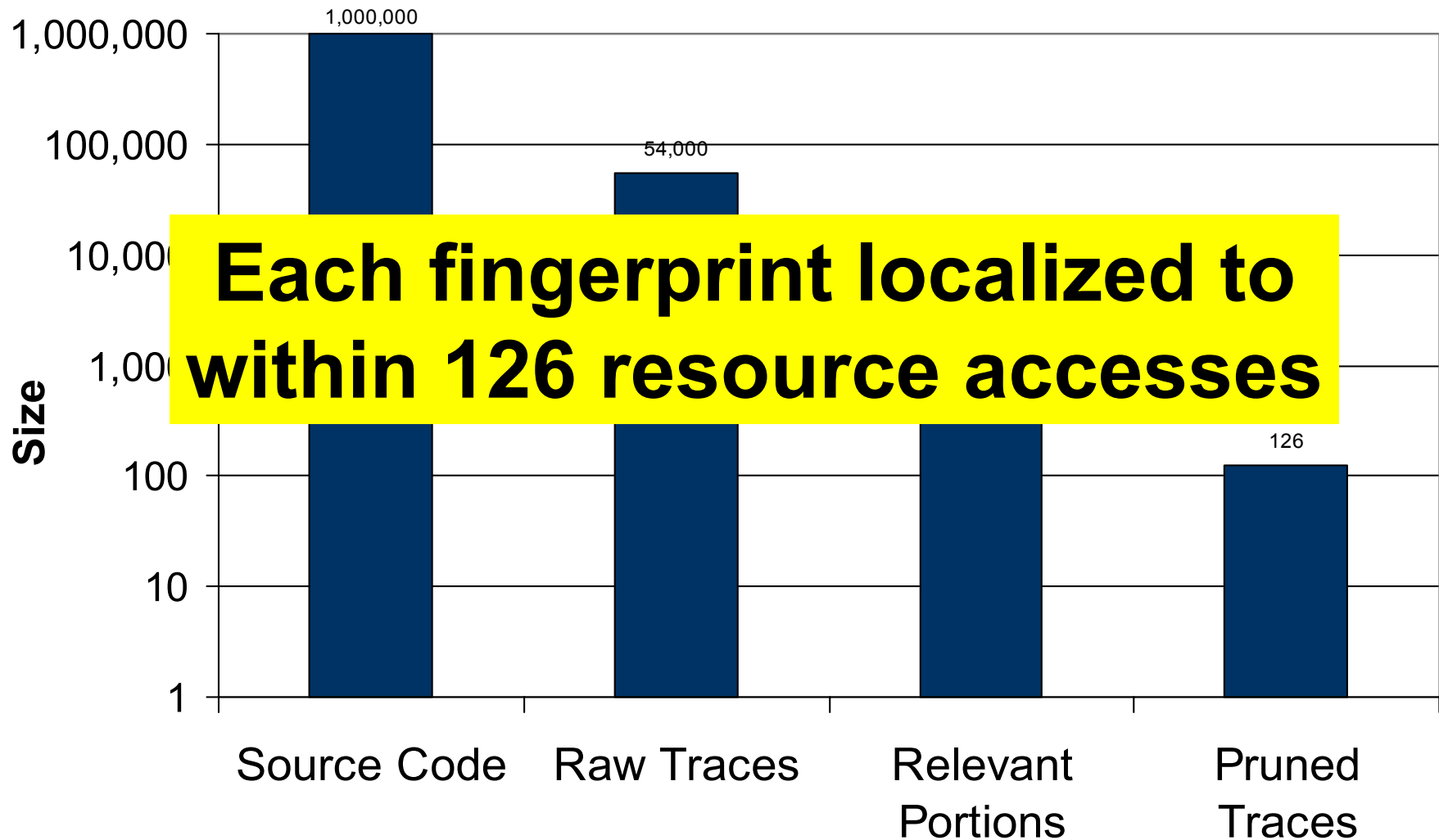
# Set equations

- Each trace has a set of labels
  - Open `xterm`: *{Create, Map}*
  - Browser: *{Create, Destroy, Map, Unmap}*
  - Move `xterm`: *{Map, Input\_Event}*
- Need set equation for *{Create}*
  - Compute an **exact cover** for this set
  - Open `xterm`  $\cap$  Open browser – Move `xterm`
- Perform the same set operations on the set of resource accesses in each trace

# Experimental methodology



# Dynamic mining: Results



# Limitations of dynamic mining

## Security-sensitive operations

***Input\_Event***

*Create*

*Destroy*

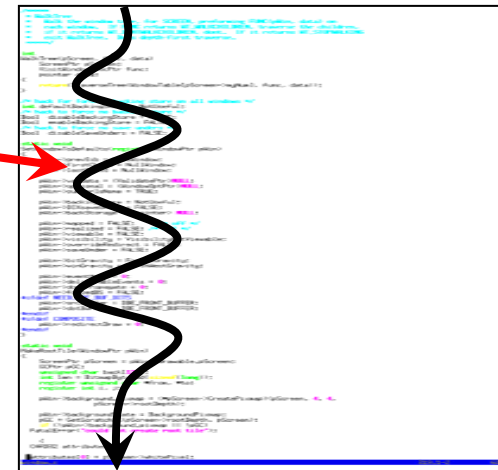
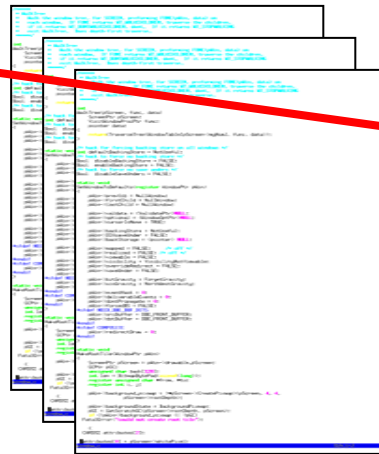
*Copy*

*Paste*

*Map*

Source Code

Runtime trace



1. Incomplete: False negatives
2. High-level description needed
3. Operations are manually induced

# Outline

- Motivation
- Problem
- **Solution**
  - Fingerprints
  - Dynamic fingerprint mining
  - **Static fingerprint mining**

**[ICSE'07]**

# Static fingerprint mining

## Security-sensitive operations

*Input\_Event*  
*Create*  
*Destroy*  
*Copy*  
*Paste*  
*Map*

## Source Code



## Resources

- **Window**
- **xEvent**

**Output: Candidate Fingerprints**  
***Cmp* xEvent->type == KeyPress**

# Problem definition

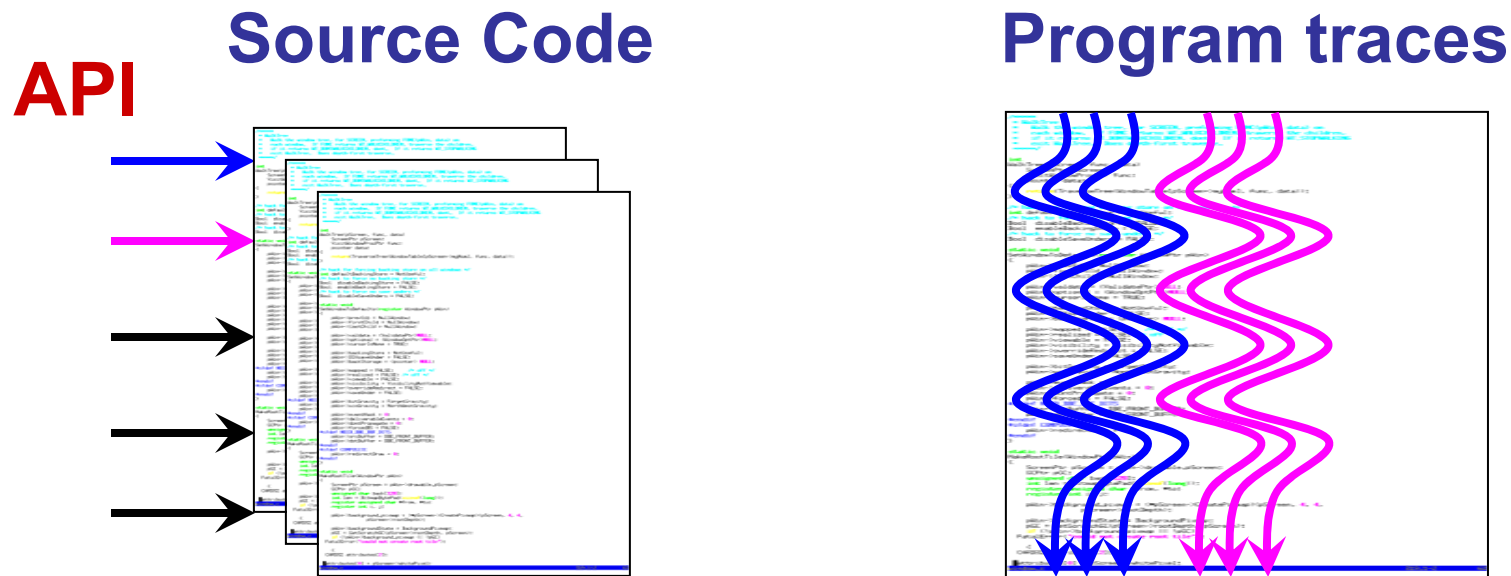
- **R**: Set of resource accesses
  - *Read/Set/Cmp* of `Window/xEvent`
- Each trace of the program contains a set of resource accesses from **R**
- **Problem**: Compute smallest mutually disjoint partition  $\mathbf{P} = \{C_1, C_2, \dots, C_n\}$  of **R**
  - $\mathbf{R} = C_1 \cup C_2 \cup \dots \cup C_n$
  - Resource accesses in each trace of the program are composed of elements of **P**

# Problem definition



- $C_1, C_2, \dots, C_n$  called **candidate fingerprints**
- **Hypothesis**: Candidate fingerprints represent security-sensitive operations

# Entry points define traces



- Each entry point implicitly defines a set of traces through the program
- Resource accesses performed by these traces can be statically characterized

# Static analysis

- Extract resource accesses potentially possible via each entry point
- Example from the X server
  - Entry point: **MapSubWindows(...)**
  - Resource accesses:
    - Set** `xEvent->type` **To** `MapNotify`
    - Set** `Window->mapped` **To** `True`
    - Read** `Window->firstChild`
    - Read** `Window->nextSib`
    - Cmp** `Window` **≠** `0`

# Resource accesses



	MapSub Windows		
<i>Set</i> <code>xEvent-&gt;type</code> <i>To</i> MapNotify	✓	✓	
<b>Identify candidate fingerprints by comparing resource accesses</b>			
<i>Read</i> <code>Window-&gt;nextStep</code>	✓		
<i>Cmp</i> <code>Window ≠ 0</code>	✓		
			✓

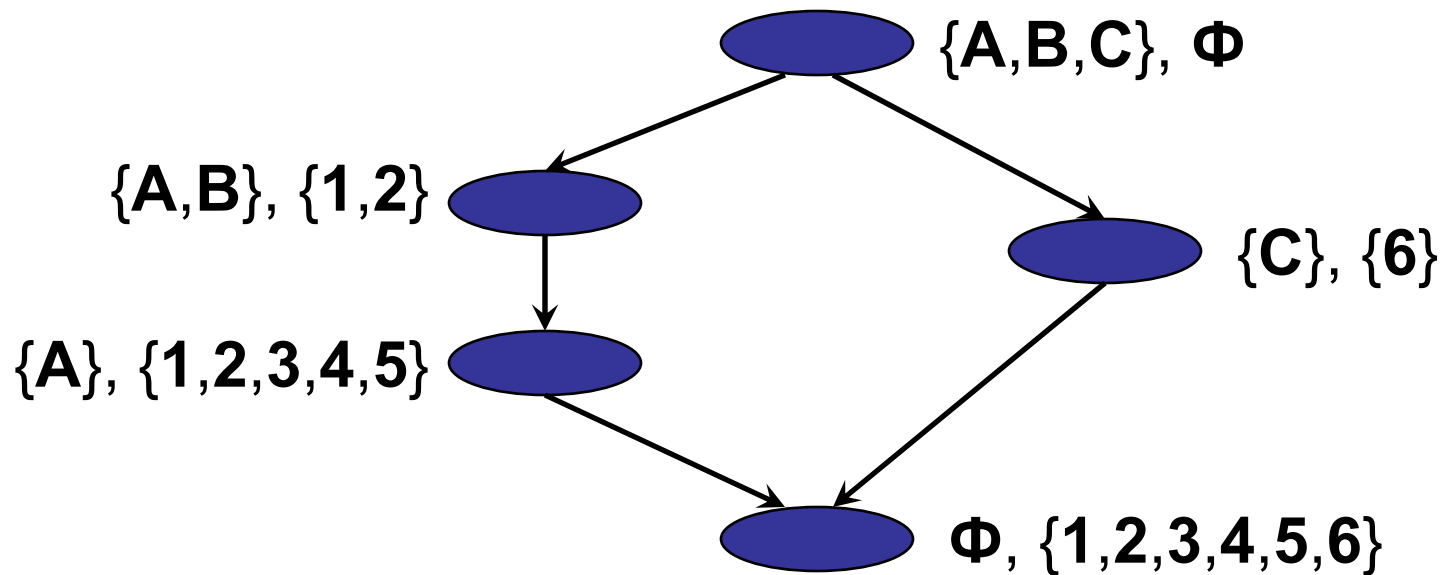
# Concept analysis



Instances	MapSub Windows	Map Window	Keyboard Input	
<i>Set</i> <code>xEvent-&gt;type</code> <i>To</i> <code>MapNotify</code>	✓	✓		
<i>Set</i> <code>Window-&gt;</code>	<h2>Comparison via hierarchical clustering</h2>			
<i>Read</i> <code>Window</code>				
<i>Read</i> <code>Window-&gt;nextSib</code>				✓
<i>Cmp</i> <code>Window ≠ 0</code>	✓			
<i>Cmp</i> <code>xEvent-&gt;type==KeyPress</code>			✓	

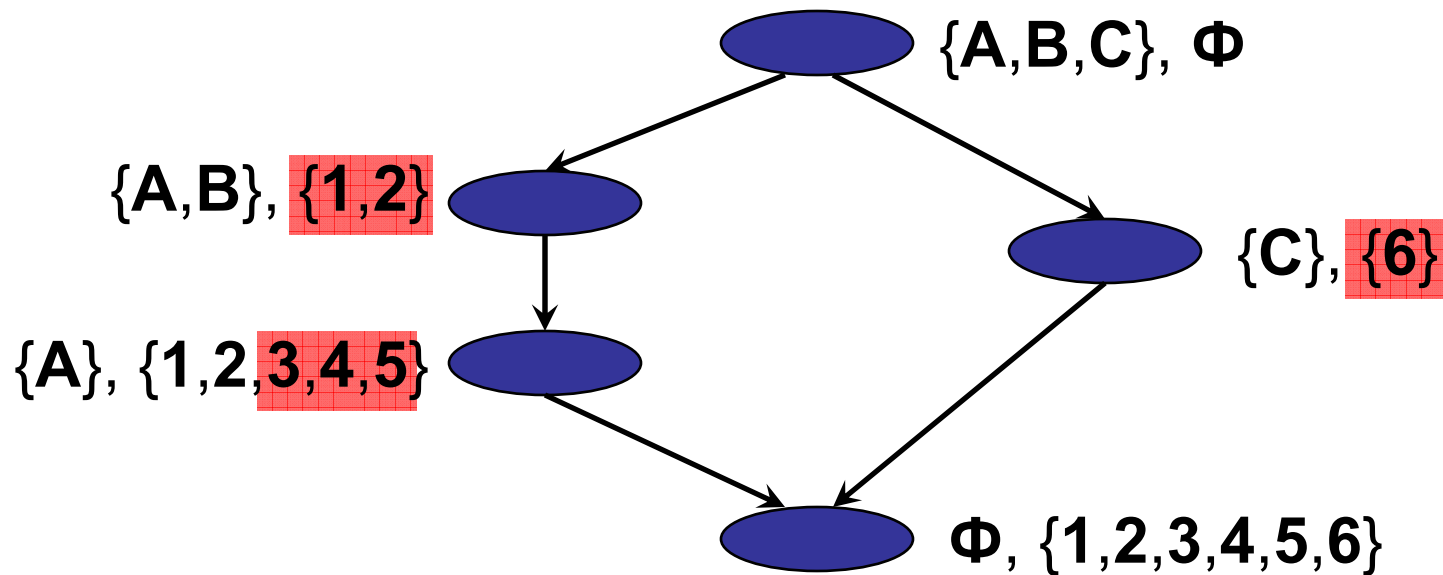
# Hierarchical clustering

		A	B	C
		MapSub Windows	Map Window	Keyboard Input
1	<i>Set</i> xEvent->type <i>To</i> MapNotify	✓	✓	
2	<i>Set</i> Window->mapped <i>To</i> True	✓	✓	
3	<i>Read</i> Window->firstChild	✓		
4	<i>Read</i> Window->nextSib	✓		
5	<i>Cmp</i> Window ≠ 0	✓		
6	<i>Cmp</i> xEvent->type==KeyPress			✓



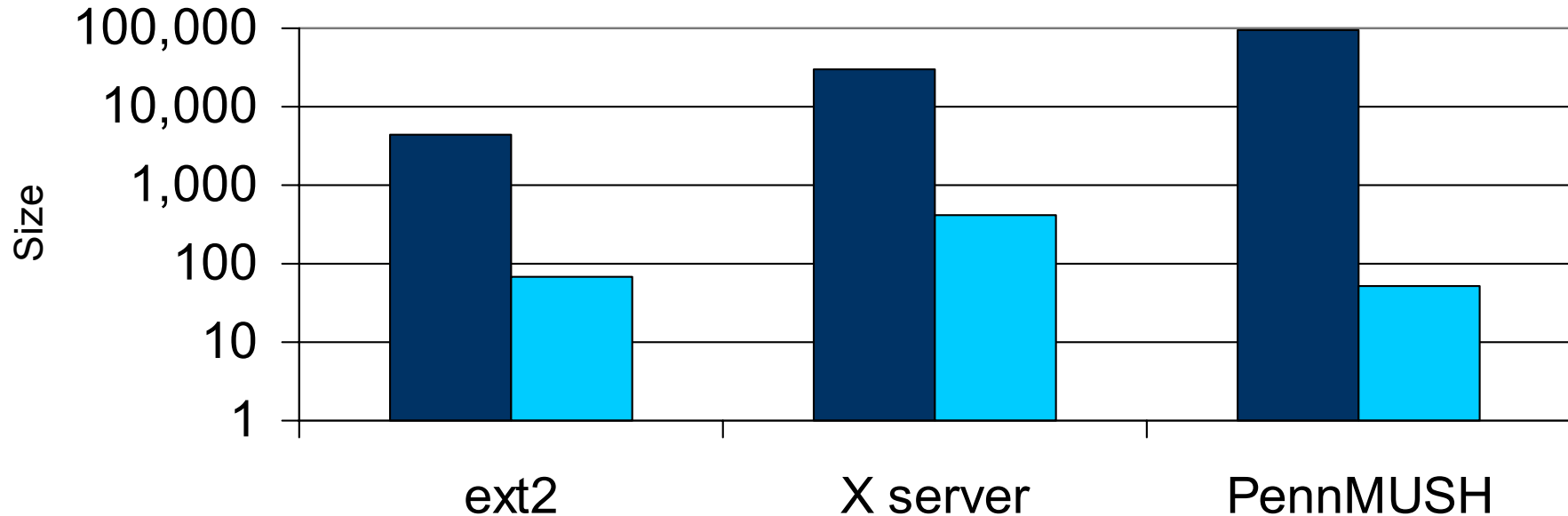
# Mining candidate fingerprints

		A	B	C
		MapSub Windows	Map Window	Keyboard Input
Cand. Fing. 1	1	Set xEvent->type To MapNotify	✓	✓
	2	Set Window->mapped To True	✓	✓
Cand. Fing. 2	3	Read Window->firstChild	✓	
	4	Read Window->nextSib	✓	
Cand. Fing. 3	5	Cmp Window ≠ 0	✓	
	6	Cmp xEvent->type==KeyPress		✓



# Static mining: Results

<b>Benchmark</b>	<b>LOC</b>	<b>Cand. Fing.</b>	<b>Avg. Size</b>
ext2	4,476	18	3.7
X Server/dix	30,096	115	3.7
PennMUSH	94,014	38	1.4



# Static mining: Results

<b>Benchmark</b>	<b>Manually identified Security-sensitive ops</b>	<b>Candidate fingerprints</b>
ext2	11	18
X Server/dix	22	115

Able to find **at least one fingerprint** for each security-sensitive operation

# Static mining: Results

Benchmark	Manually identified Security-sensitive ops	Candidate fingerprints
ext2	11	18
X Server/dix	22	115



Identified as part of **v minutes**  
Interpreted as **w hours**  
**multi-year efforts**

# Static mining: Results

Benchmark	Manually identified Security-sensitive ops	Candidate fingerprints
ext2	11	18
X Server/dix	22	115

- Associated 59 candidate fingerprints with security-sensitive operations
- Remaining are likely security-sensitive too
  - Read** `Window->DrawableRec->width` &
  - Read** `Window->DrawableRec->height`

# Summary of contributions

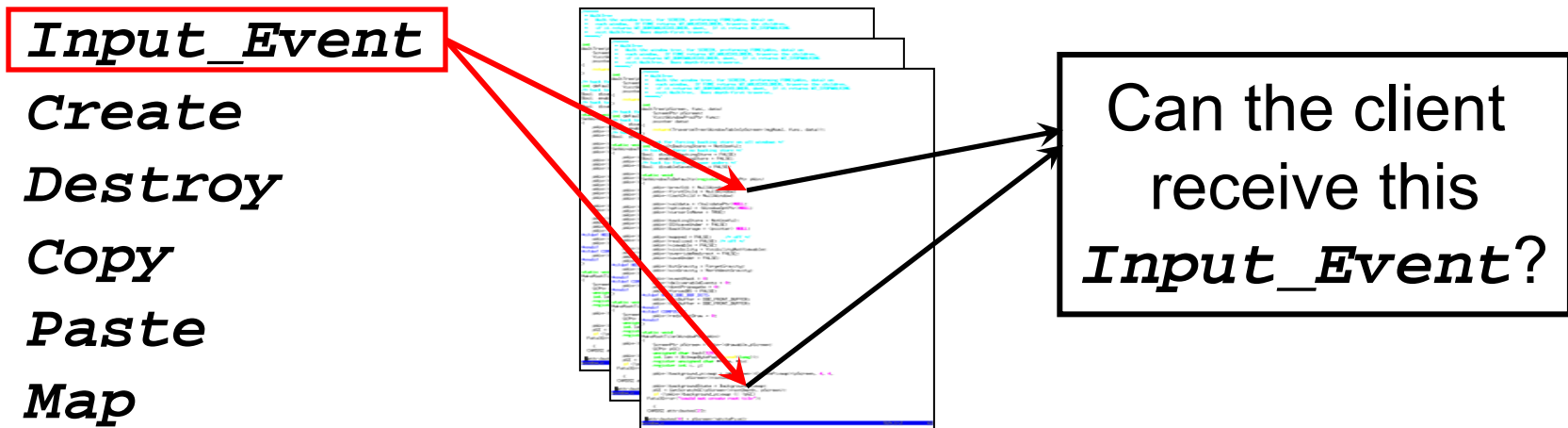
## Fingerprints

### Mining

[Oakland'06][ICSE'07]

### Matching

[CCS'05]



# Implications

## Can reduce manual effort

- **Before:** Approximately 2 years
- **After:** Few hours
  - Analysis: ~ minutes; Interpretation: ~ hours

## Can reduce errors

- **Before:** Violation of complete mediation
- **After:** Basis to prove security

# Future work

- Emitting security proofs
  - Guarantee that retrofitted code satisfies principle of complete mediation
  - Counter-example → must add additional authorization checks
- More expressive fingerprint languages
  - Temporal information on resource accesses
  - Encode dataflow facts in fingerprints

# Retrofitting Legacy Code for Security

Vinod Ganapathy

Computer Sciences Department  
University of Wisconsin-Madison

`vg@cs.wisc.edu`

`http://www.cs.wisc.edu/~vg`