Retrofitting Legacy Code for Authorization Policy Enforcement

Vinod Ganapathy

Ph.D. Thesis Defense

Thursday, July 12th, 2007

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

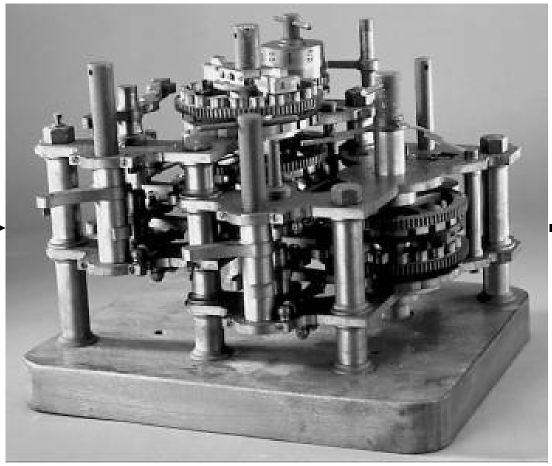
Relevance of the Principle today

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

Legacy code →  → Retrofitted code

INSECURE **SECURE**

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

Resource user

Operation request ↓ ↑ Response

Resource manager

Reference monitor

Allowed? ↓ ↑ **YES**/**NO**

⟨Alice, /etc/passwd, ***`File_Read`***⟩

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]

Thesis statement

Program analysis and transformation techniques offer a principled and automated way to retrofit legacy code with reference monitors

Contributions

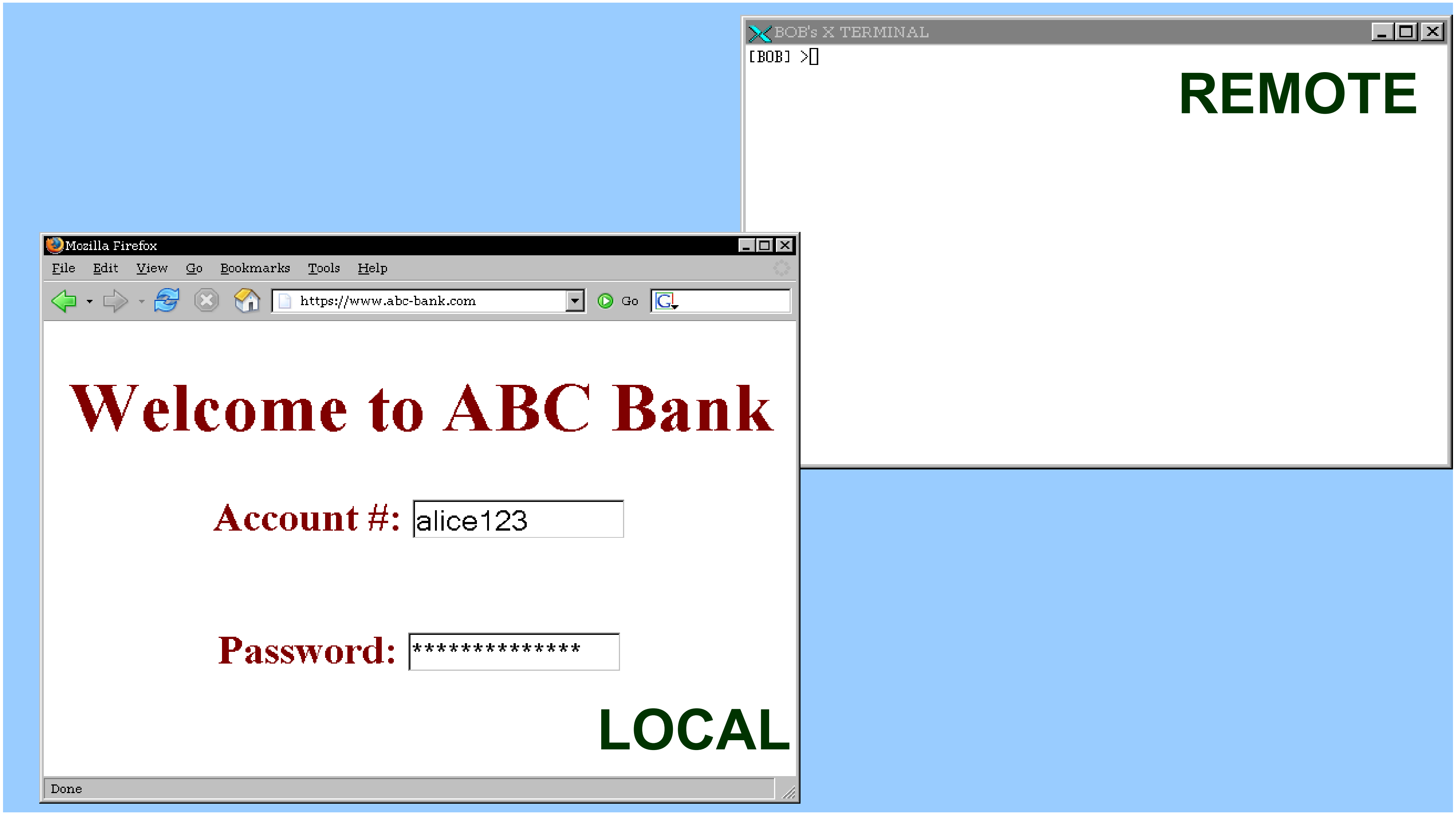
Analyses and transformations for authorization policy enforcement

- Fingerprints: A new representation for security-sensitive operations
- Two algorithms to mine fingerprints
- Result: Reduced effort to retrofit legacy code for authorization policy enforcement
  - Manual effort needed reduces to a few hours
  - Applied to X server, Linux kernel, PennMUSH

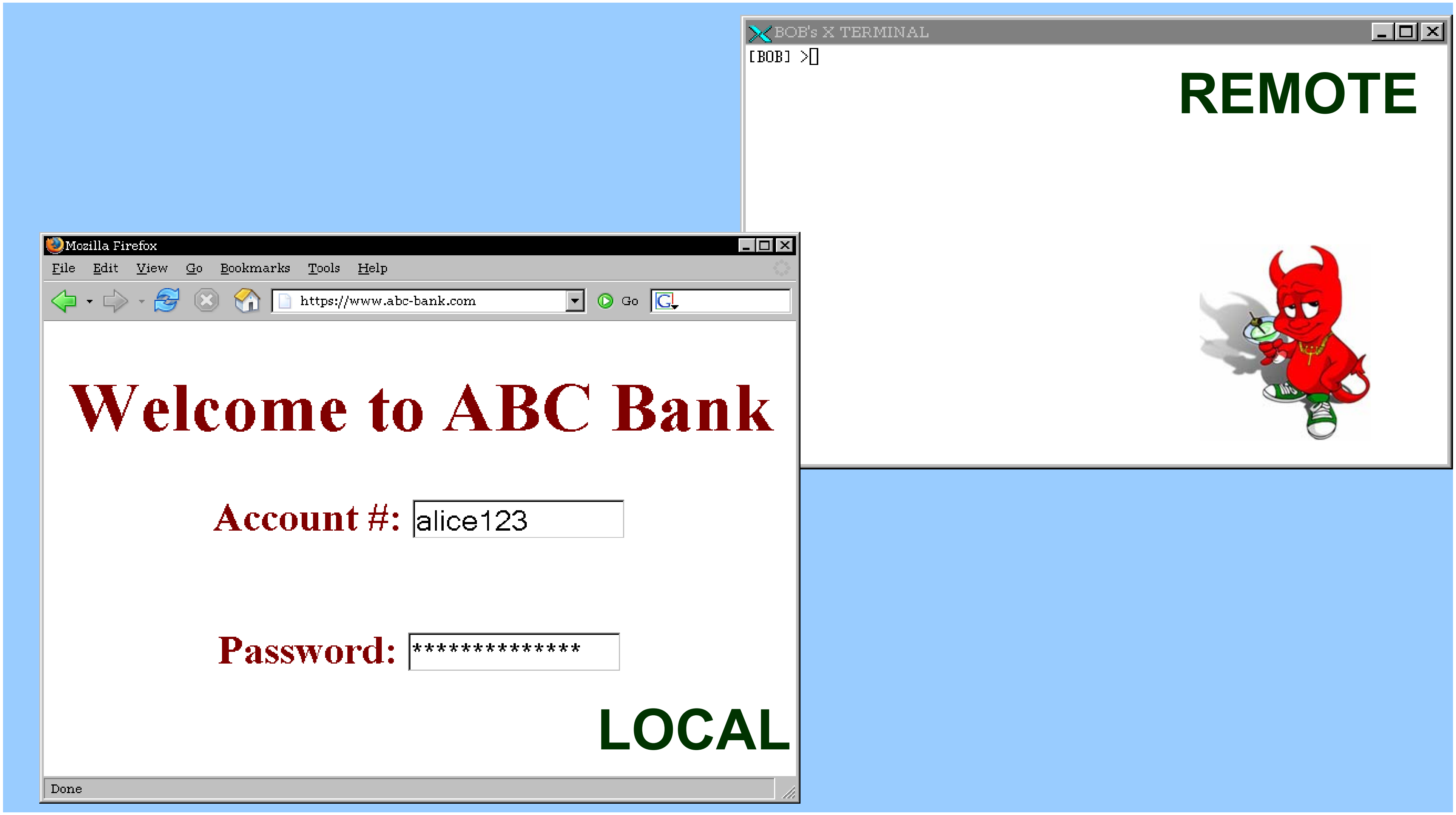
Outline

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

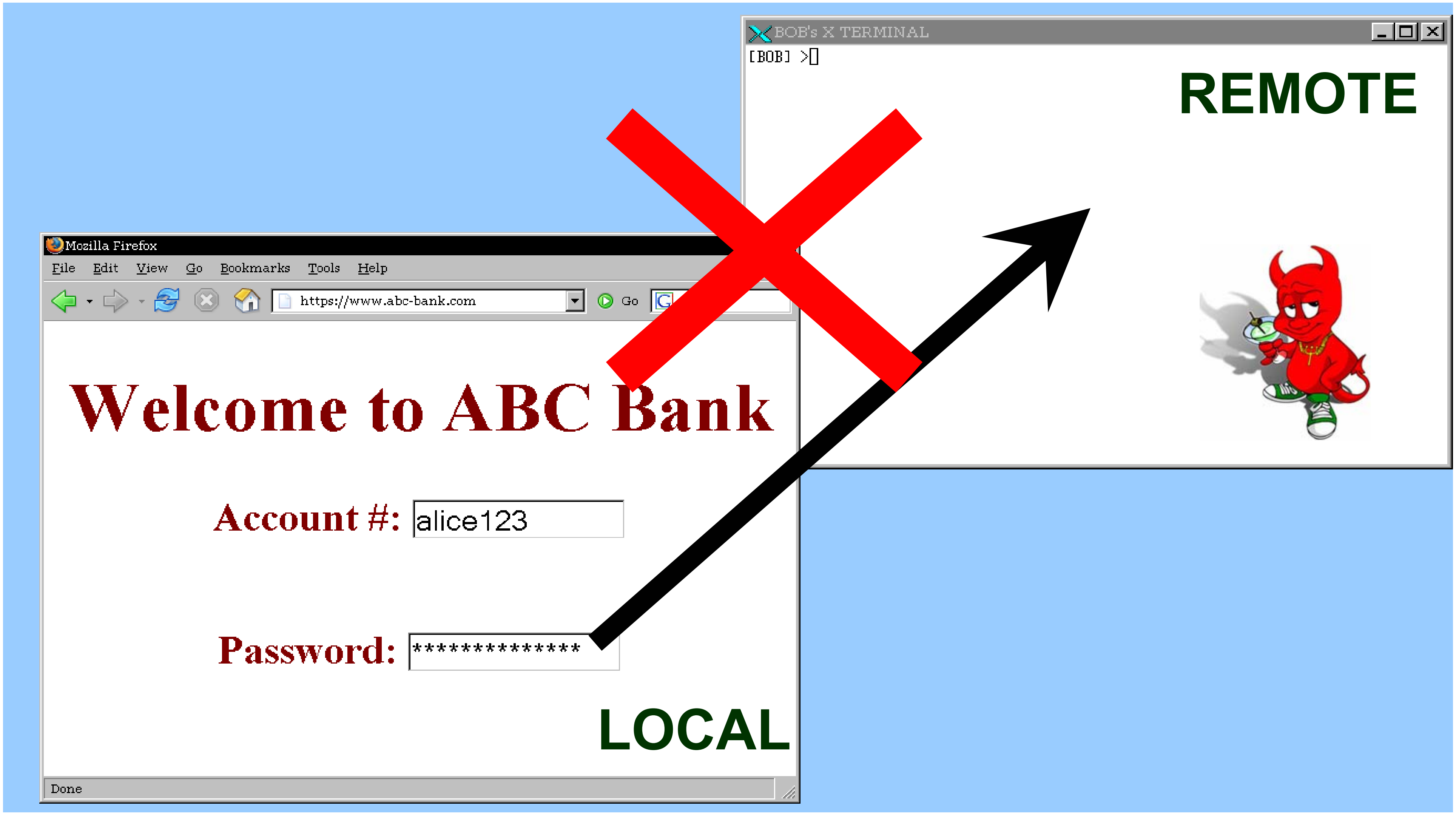
X server with multiple X clients



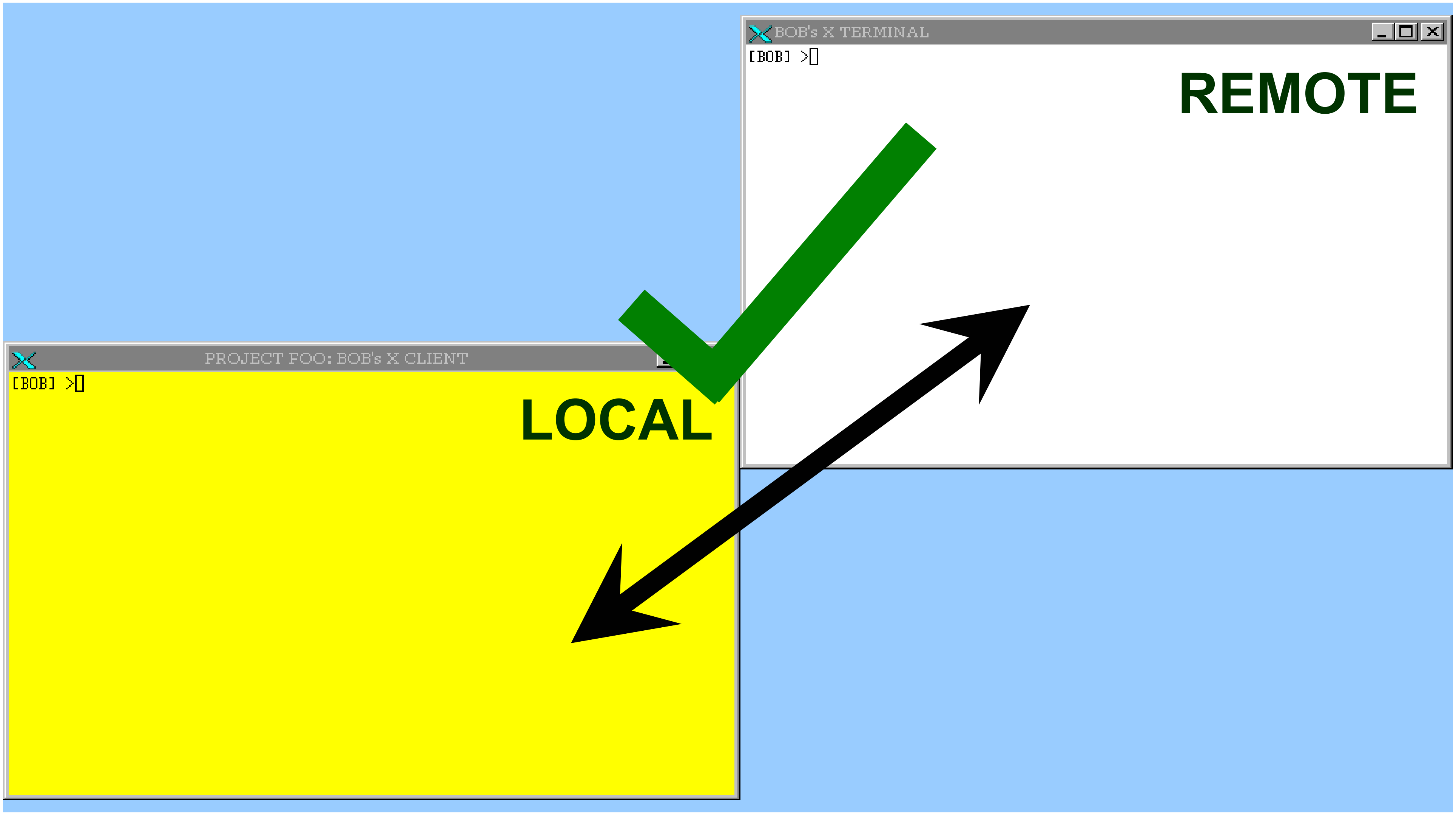
Malicious remote X client



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

X client

Operation request

Response

X server

Reference monitor

Allowed?

YES/**NO**

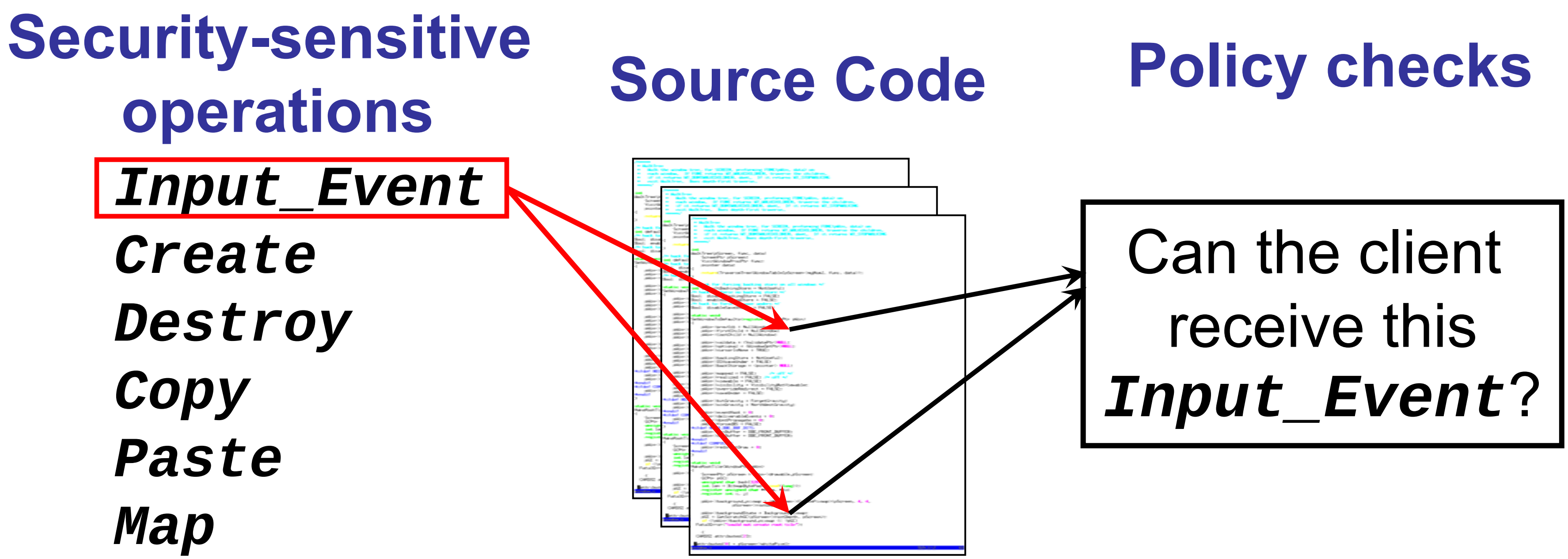
Authorization policy

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



Problems

Manual

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

Ad hoc

- Violation of complete mediation
- Time-of-check to Time-of-use bugs [Zhang *et al.*, ‘02][Jaeger *et al.*, ‘04]

Our approach

Principled

- Fingerprints: A new representation of security-sensitive operations

Automated

- Legacy code retrofitted using fingerprints
  - Use of static and dynamic program analysis

Approach overview

Legacy code

↓

Miner

↓

Fingerprints

↓

Matcher

↓

Retrofitted code

Outline

- Motivation
- Problem
- Solution
  - Fingerprints **[CCS'05]**
  - Dynamic fingerprint mining
  - Static fingerprint mining

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

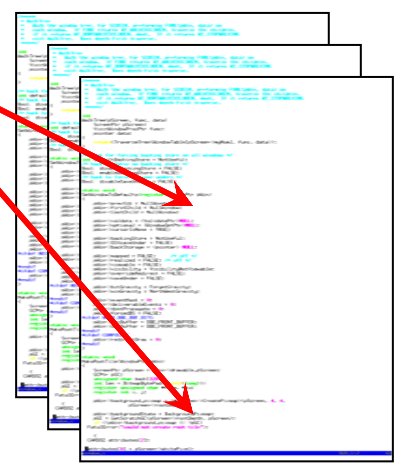
- ***`Input_Event :-`***

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

Security-sensitive operations

Source Code

`Input_Event`
`Create`
`Destroy`
`Copy`
`Paste`
`Map`



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 [Oakland'06]
  - Static fingerprint mining

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]

`Input_Event`	Input to window from device
`Create`	Create new window
`Destroy`	Destroy existing window
`Map`	Map window to console

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

Problem definition

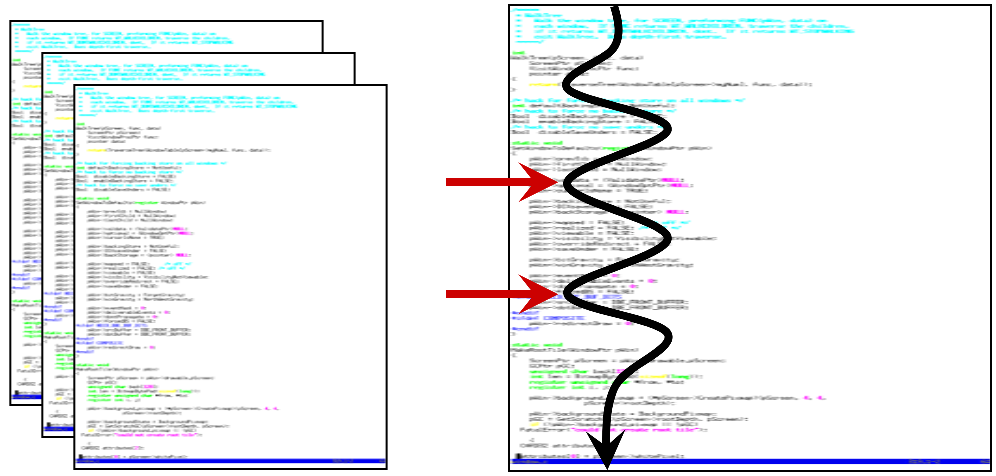
- **S**: Set of security-sensitive operations
- **D**: Descriptions of operations in **S**
- **R**: Set of resource accesses
  - ***Read***/***Set***/***Cmp*** of **`Window`**/**`xEvent`**
- Each **`s`** є **S** has a fingerprint
  - A fingerprint is a subset of **R**
  - Contains a resource access unique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

Traces contain fingerprints

Security-sensitive operations

Source Code **Runtime trace**

`Input_Event`
`Create`
`Destroy`
`Copy`
`Paste`
`Map`



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

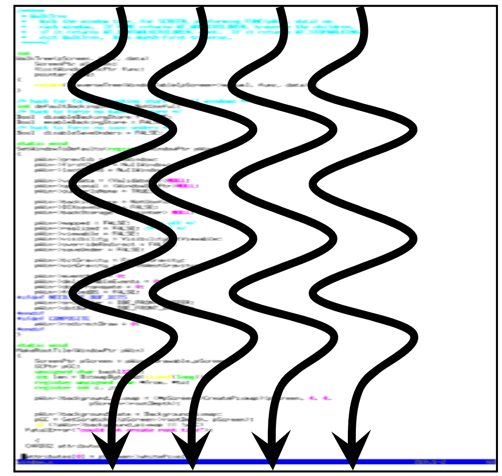
Compare traces to localize

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
`Create`	✓			✓	
`Destroy`		✓		✓	
`Map`	✓		✓	✓	
`Unmap`		✓		✓	
`Input_Event`			✓		✓

Comparison with "diff" and "∩"

Perform same set operations on resource accesses

	Open `xterm`	Close `xterm`	Move `xterm`	Open browser	Switch windows
Create	✓			✓	
Destroy		✓		✓	
Map	✓		✓	✓	
Unmap		✓		✓	
Input_Event			✓		✓

Create = Open **xterm** ∩ 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`** ∩ Open browser – Move **`xterm`**
- Perform the same set operations on the set of resource accesses in each trace

Experimental methodology

Source code

`gcc --enable-logging`

Server with logging enabled

Run experiments and collect traces

Raw traces

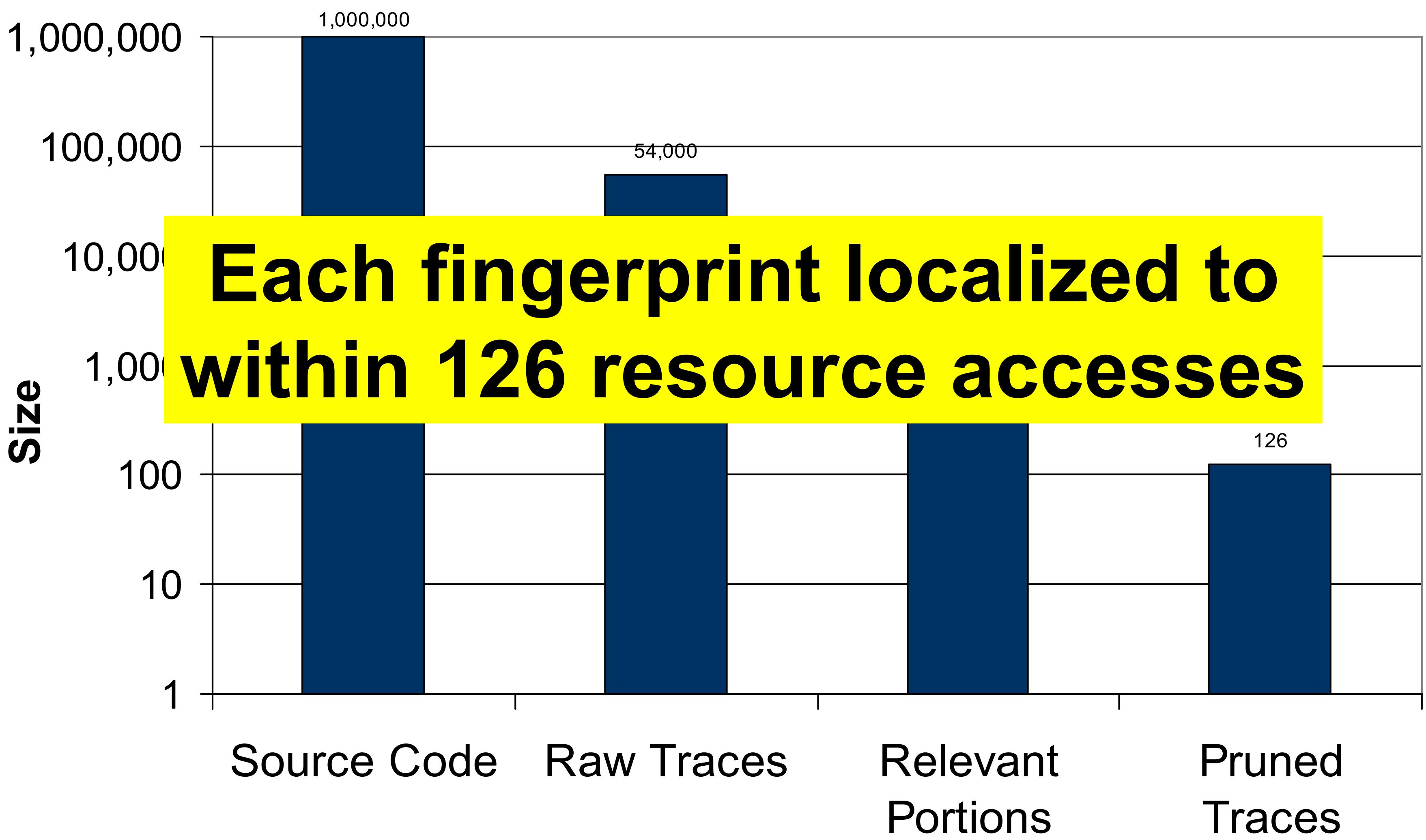
Localize security-sensitive operation

Relevant portions of traces

Compare traces with “diff” and “∩”

Pruned traces

Dynamic mining: Results

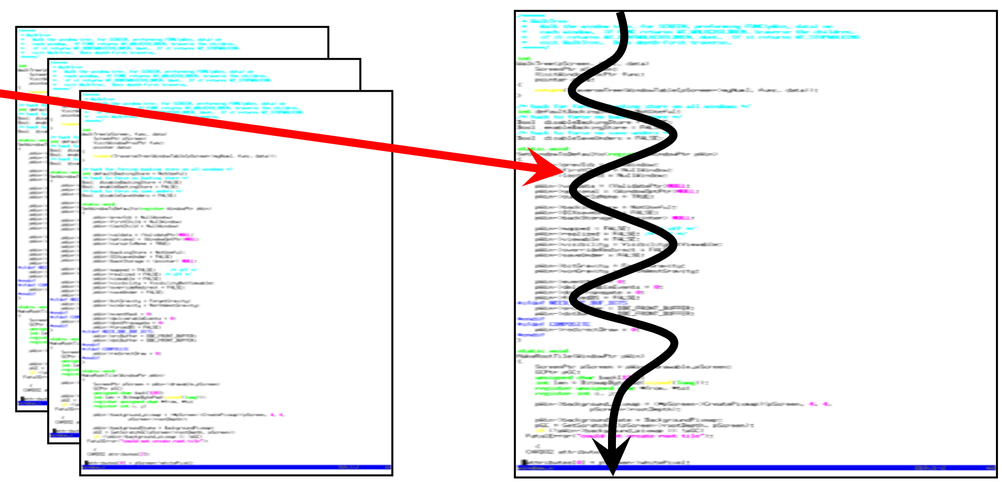


Each fingerprint localized to within 126 resource accesses

Limitations of dynamic mining

Security-sensitive operations

Input_Event
Create
Destroy
Copy
Paste
Map



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`**
- **E**: Set of entry points into the server
- **Goal**: Find fingerprints using **R** and **E**

Not given an *a priori* description of security-sensitive operations

Straw-man proposal I

Each resource access in R is a fingerprint

- Finest level of granularity
- ***Cmp*** **`xEvent->type == KeyPress`**
- ***Read*** **`Window->firstChild`**
- ***Read*** **`Window->nextSib`**
- ***Cmp*** **`Window ≠ 0`**

Problem with this proposal

Difficult to write and maintain policies at this level of granularity

- ***Cmp*** **`xEvent->type == KeyPress`**
- ***Read*** **`Window->firstChild`**
- ***Read*** **`Window->nextSib`**
- ***Cmp*** **`Window ≠ 0`**

Straw-man proposal II

Each API in E is a fingerprint

- Coarsest level of granularity

- ***Call*** **`MapSubWindows`**
- ***Call*** **`MapWindow`**

- Write policies allowing/disallowing the use of an API call

Problem with this proposal

Does not reflect actual resource accesses performed by API call

- ***Call*** **`MapSubWindows`**
  - Enumerates child windows and maps them to the screen
- ***Call*** **`MapWindows`**
  - Maps a window onto the screen

Our approach

Cluster resource accesses that always happen together

- Each API entry point implicitly defines a set of resource accesses
- Cluster resource accesses based upon the API entry points that perform them

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		
Set **`xEvent->type`** ***To*** MapNotify	✓	✓	
Se			
Re			
Read **`Window->nextSib`**	✓		
Cmp **`Window ≠ 0`**	✓		
			✓

Identify candidate fingerprints by clustering resource accesses

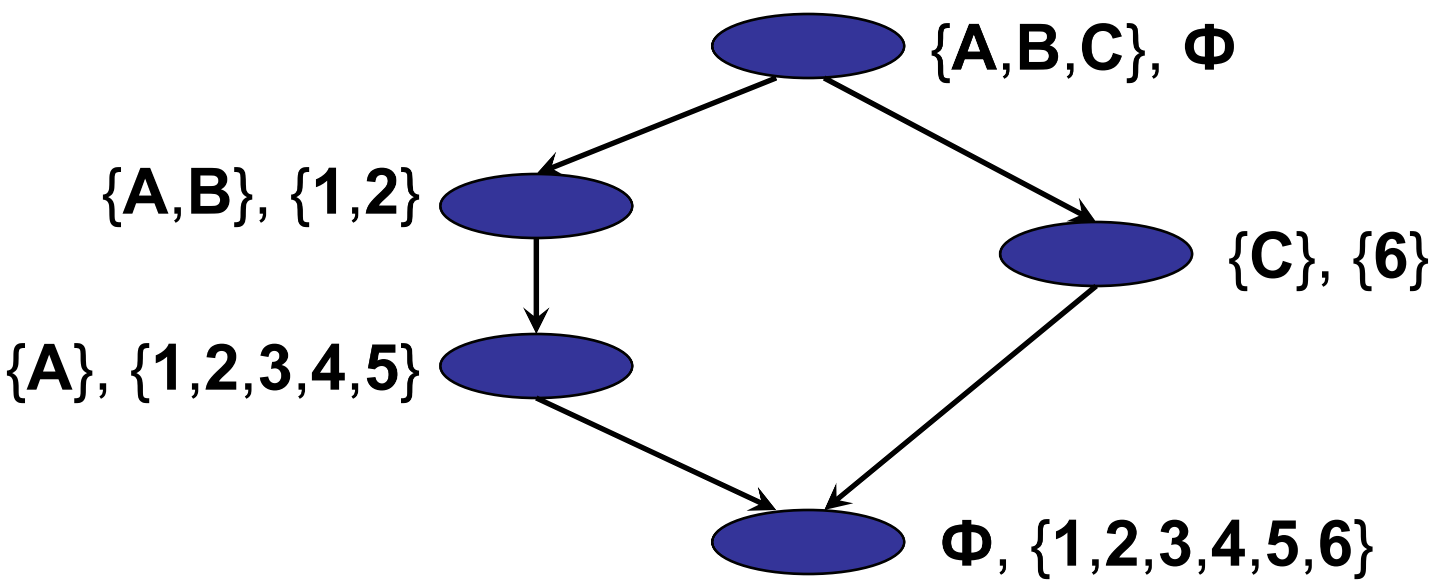
Concept analysis

Instances	MapSub Windows	Map Window	Keyboard Input
Set `xEvent->type` *To* MapNotify	✓	✓	
Set `Window-`			
Read `Window`			
Read `Window->nextSib`	✓		
Cmp `Window ≠ 0`	✓		
Cmp `xEvent->type==`KeyPress			✓

Comparison via hierarchical clustering

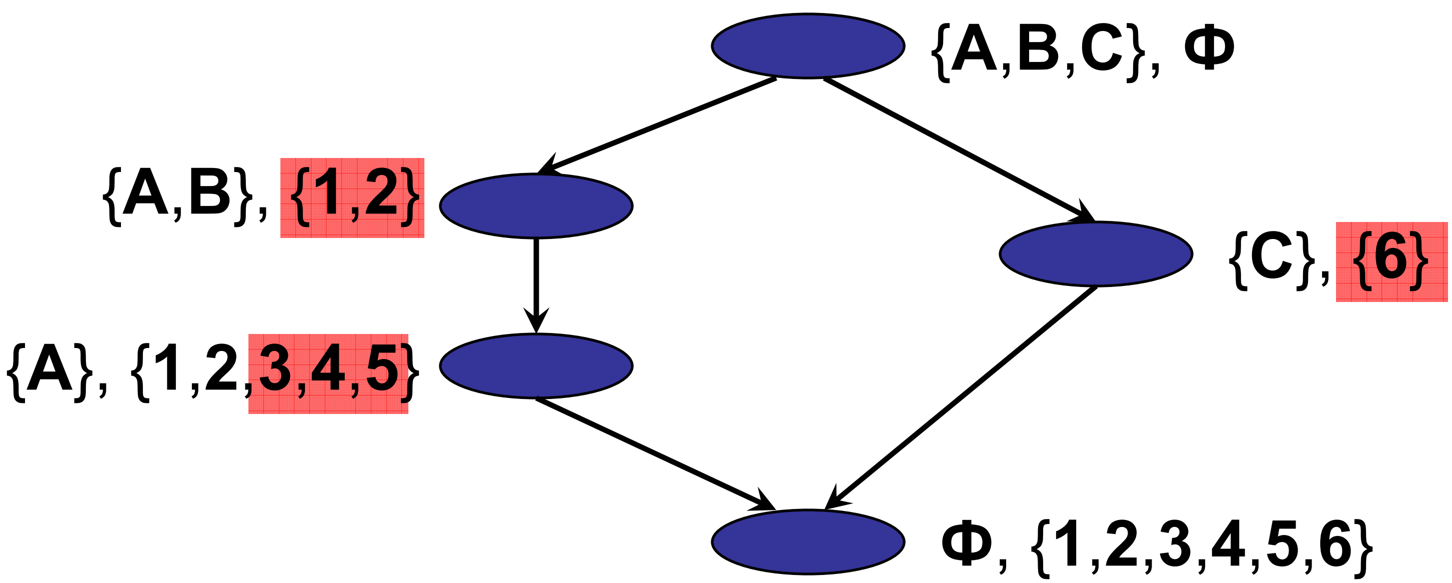
Hierarchical clustering

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



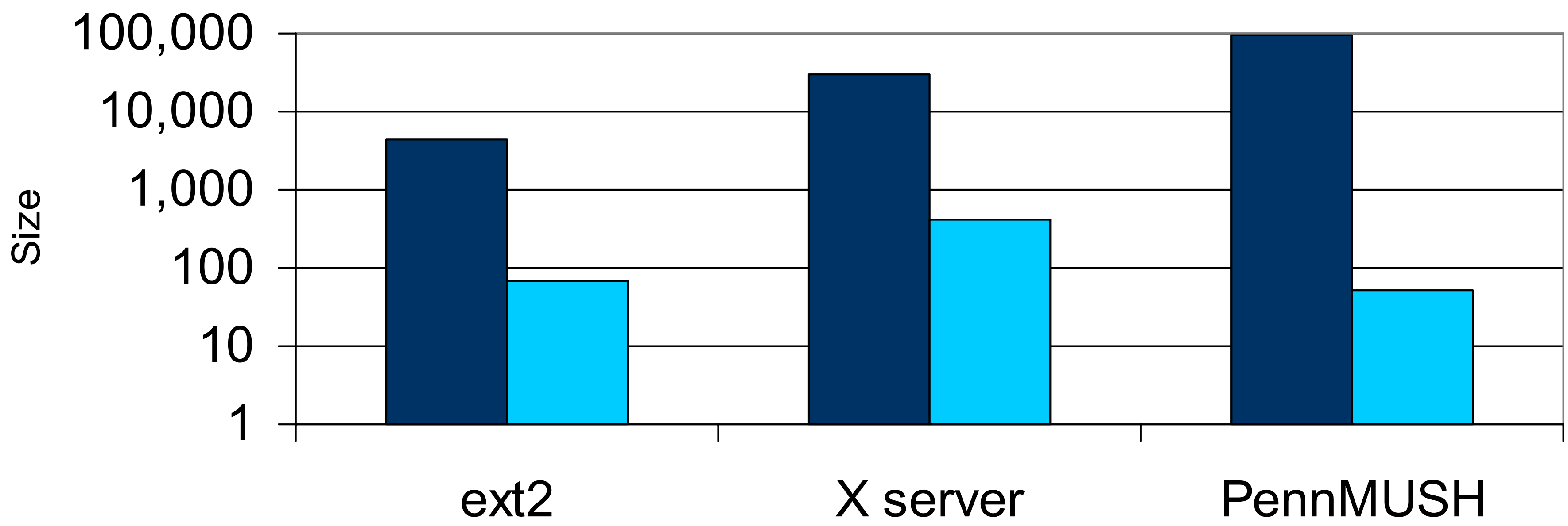
Mining candidate fingerprints

			A MapSub Windows	B Map Window	C 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`	✓		
	5	*Cmp* `Window ≠ 0`	✓		
Cand. Fing. 3	6	*Cmp* `xEvent->type==`KeyPress			✓



Static mining: Results

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



Static mining: Results

Benchmark	Manually identified Security-sensitive ops	Candidate fingerprints
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 Identified as part of v minutes
Interpre **multi-year efforts** w hours

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

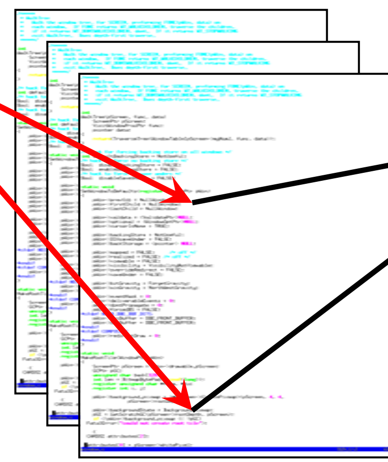
  Read **`Window->DrawableRec->width`** **&**
  Read **`Window->DrawableRec->height`**

Summary of contributions

Fingerprints

Mining | **Matching**

Input_Event
Create
Destroy
Copy
Paste
Map



Can the client receive this ***Input_Event***?

Lessons for the future

Modifying legacy code is non-trivial

- Modifications may break software
- Modifying executables is challenging

Low-overhead runtime system for policy enforcement on unmodified code

Lessons for the future

Soundness/completeness hard to achieve for C

- Type-safety violations the main problem

Provable guarantees with additional runtime checks?

Lessons for the future

Difficult to automate failure handling

- Failure handling is a crosscutting-concern
- Handling failure gracefully is the main challenge

Aspect-oriented solution?

Checkpoint and rollback?


"That's all Folks!"

Errors in labeling traces (I)

	Open `xterm`	Close `xterm`	Move `xterm`	Open browser	Switch windows
`CREATE`	✓			✓	
`DESTROY`		✓		✓	
`MAP`	✓		✓	✓	
`UNMAP`		✓		✓	
`INPUTEVENT`			✓		✓

Errors in labeling traces (I)

	Open `xterm`	Close `xterm`	Move `xterm`	Open browser	Switch windows
`CREATE`	✓				
`DESTROY`		✓		✓	
`MAP`	✓		✓	✓	
`UNMAP`		✓		✓	
`INPUTEVENT`			✓		✓

`CREATE` = Trace1 – Trace3

Errors in labeling traces (II)

	Open `xterm`	Close `xterm`	Move `xterm`	Open browser	Switch windows
`CREATE`	✓		✓	✓	
`DESTROY`		✓		✓	
`MAP`	✓		✓	✓	
`UNMAP`		✓		✓	
`INPUTEVENT`			✓		✓

Dealing with errors in labeling

- Missing labels from traces:
  - "∩" operation will not discard fingerprint
  - "diff" operation may erroneously eliminate a fingerprint
- Extra labels on traces:
  - May erroneously eliminate a fingerprint
- Trial-and-error
  - Relabel and recompute set-equations
- Empirically: tolerance of about 15% errors