Retrofitting Legacy Code for Authorization Policy Enforcement

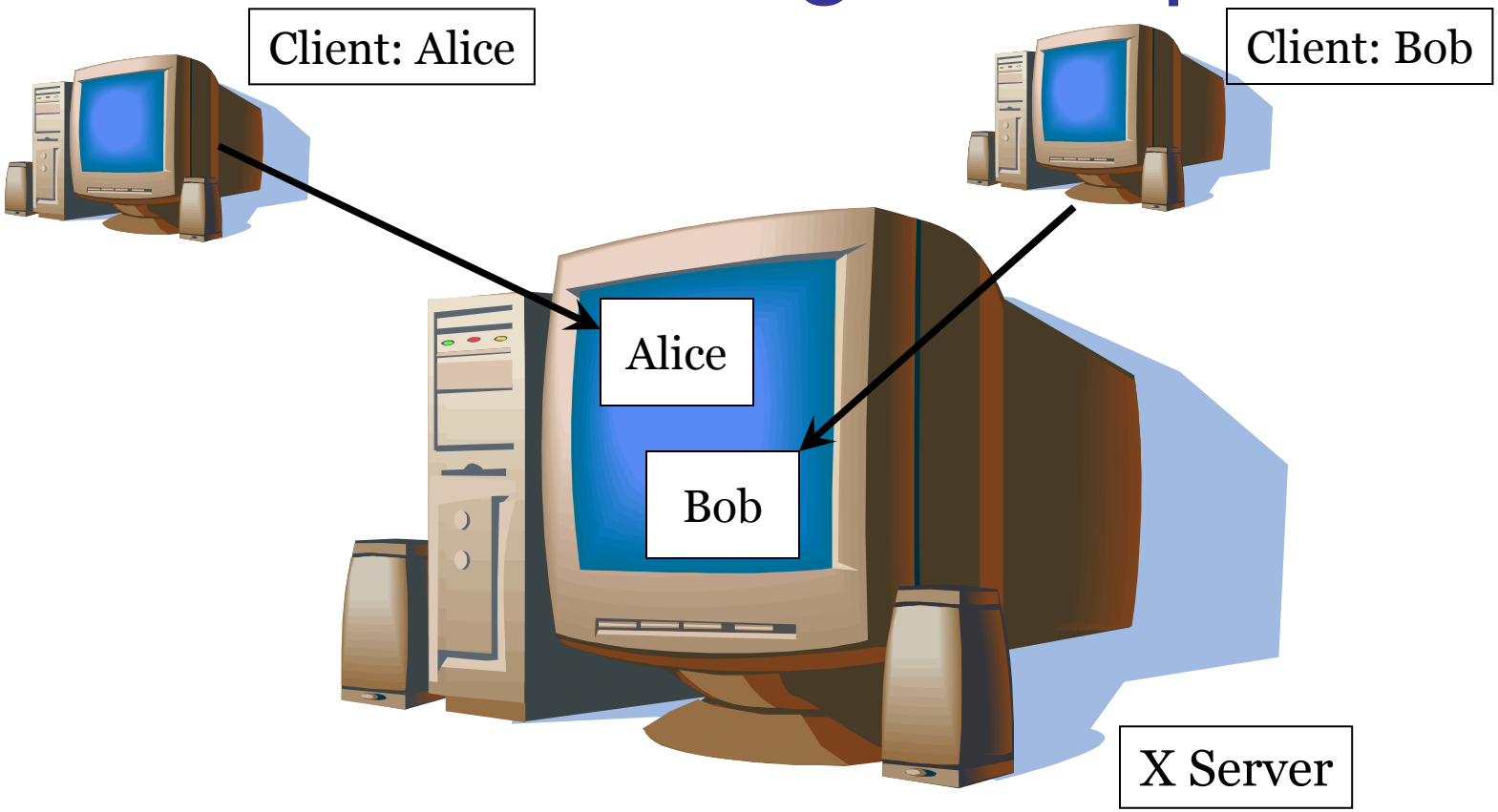
Vinod Ganapathy
vg@cs.wisc.edu

Trent Jaeger
tjaeger@cse.psu.edu

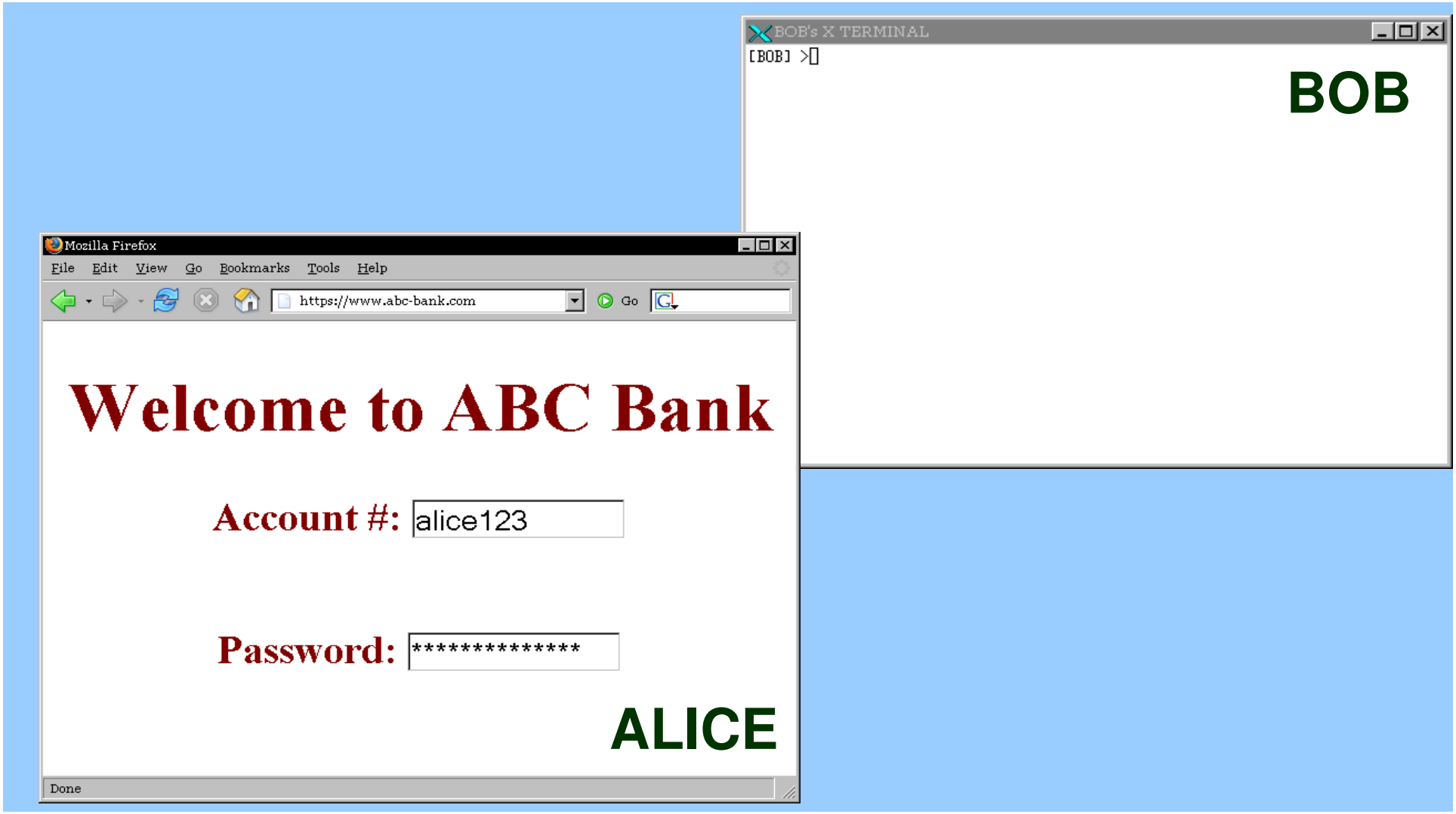
Somesh Jha
jha@cs.wisc.edu

2006 IEEE Symposium on Security and Privacy
Oakland, California

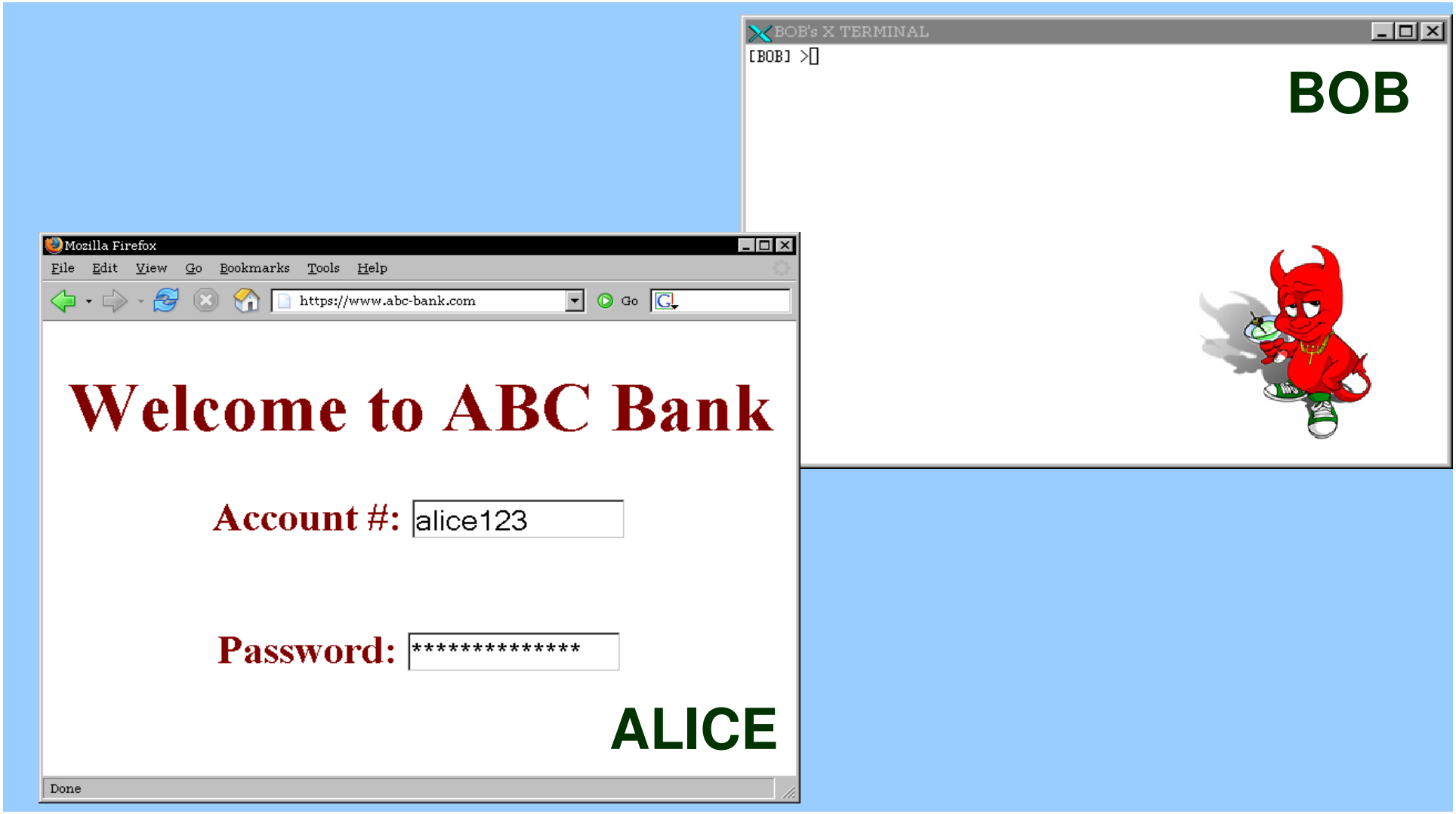
Motivating example



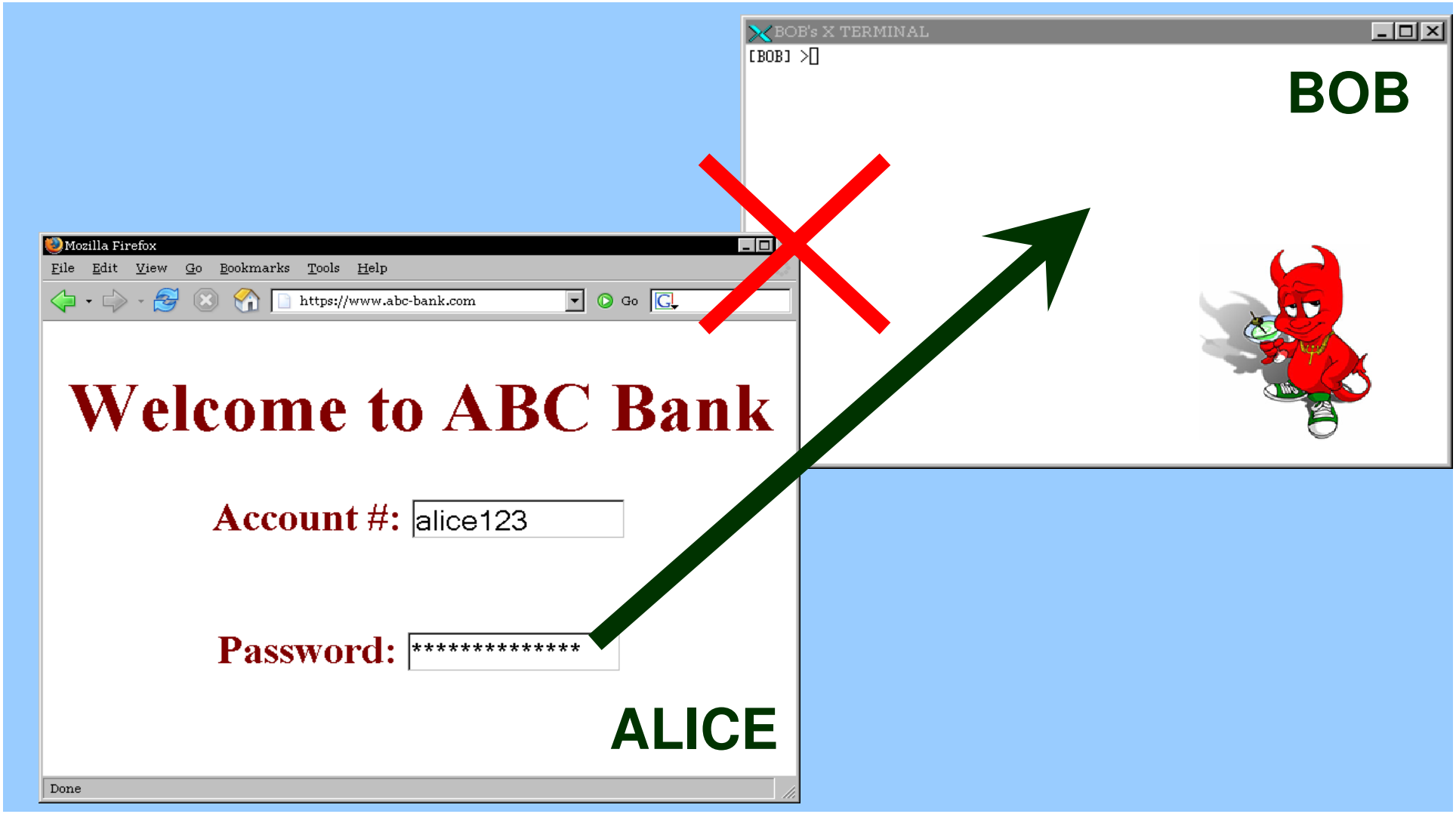
X server with multiple X clients



Bob's malicious X client

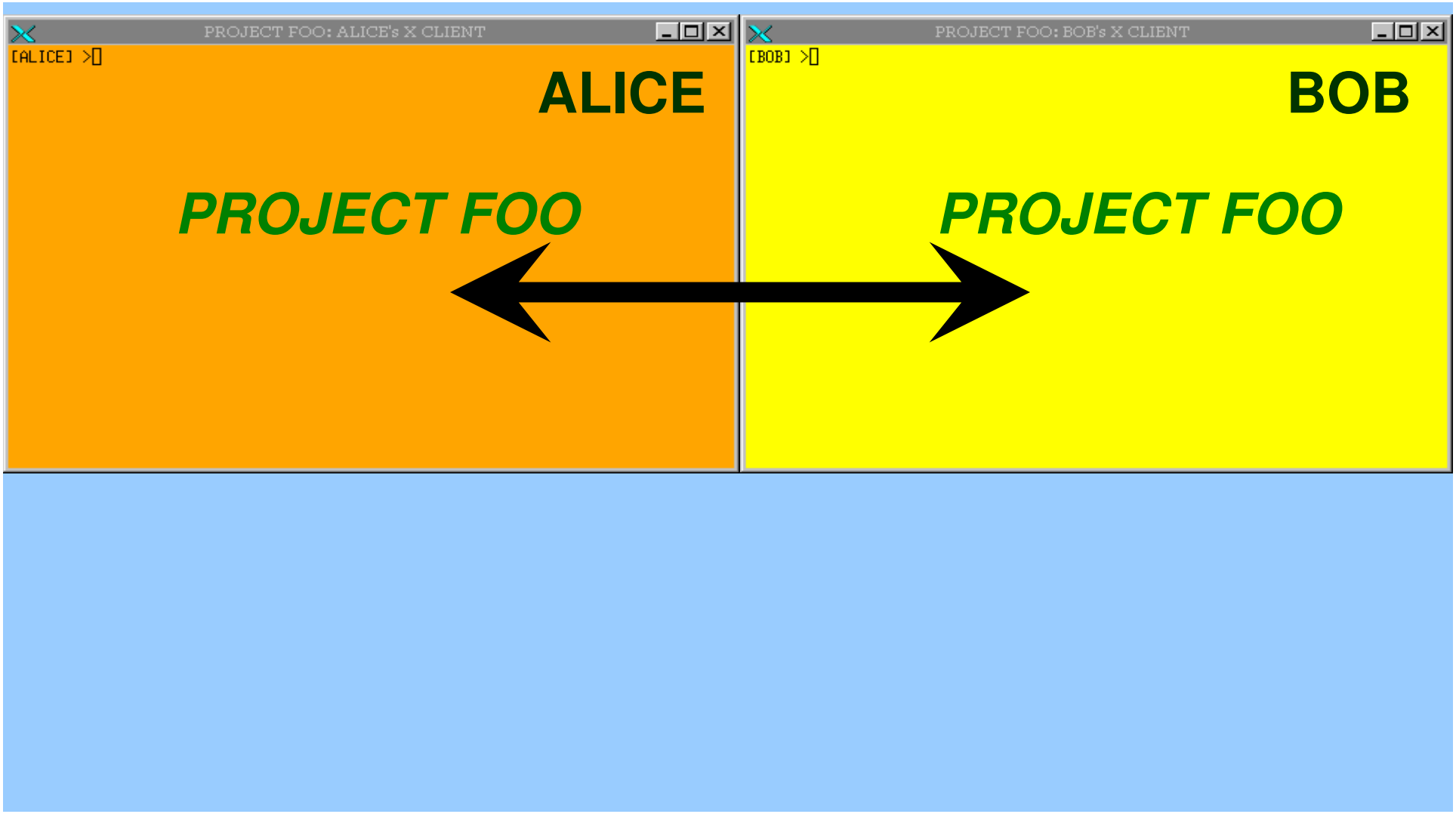


Bob stealing Alice's password

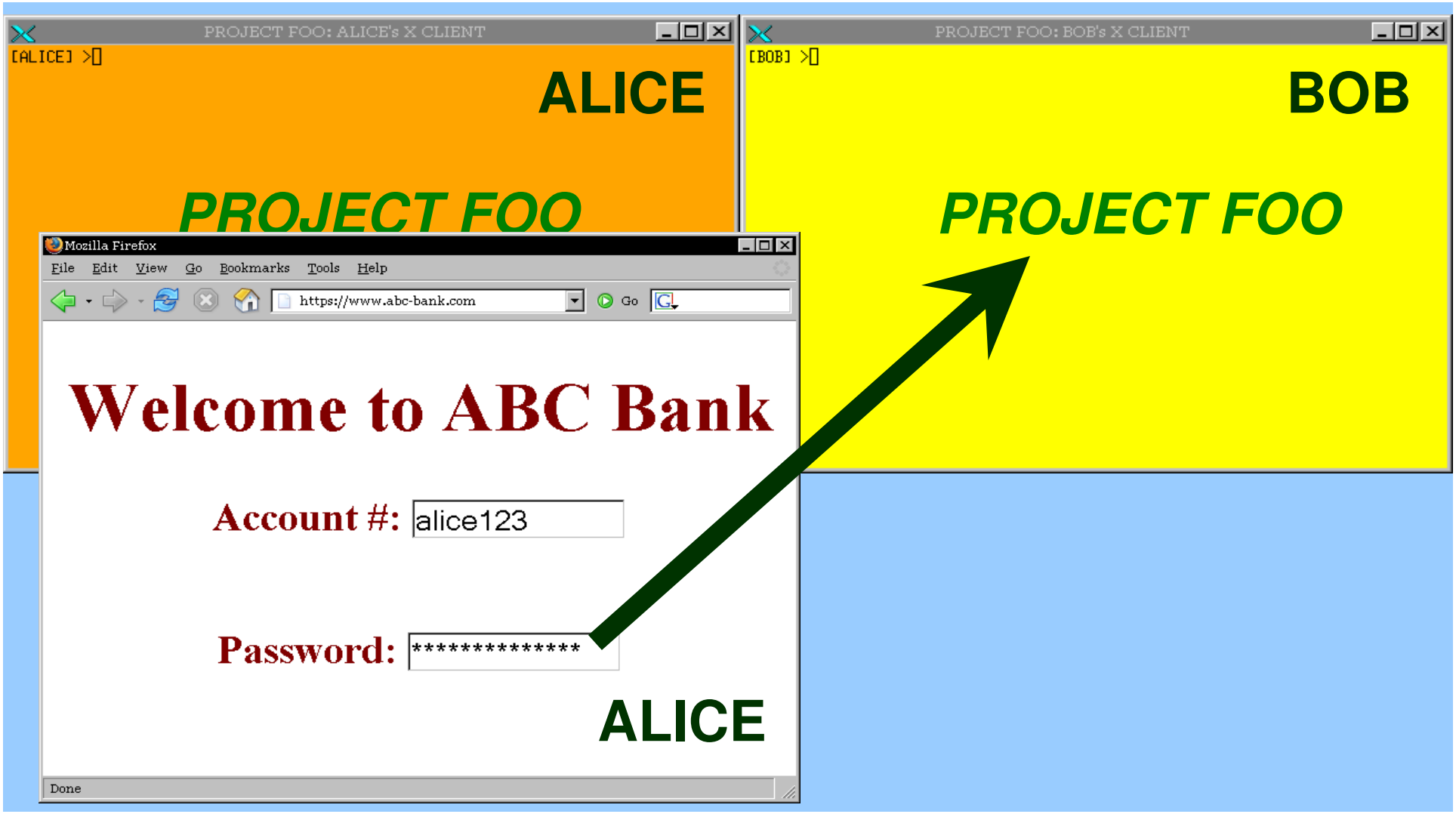


Collaboration

Desirable information flow



Undesirable information flow



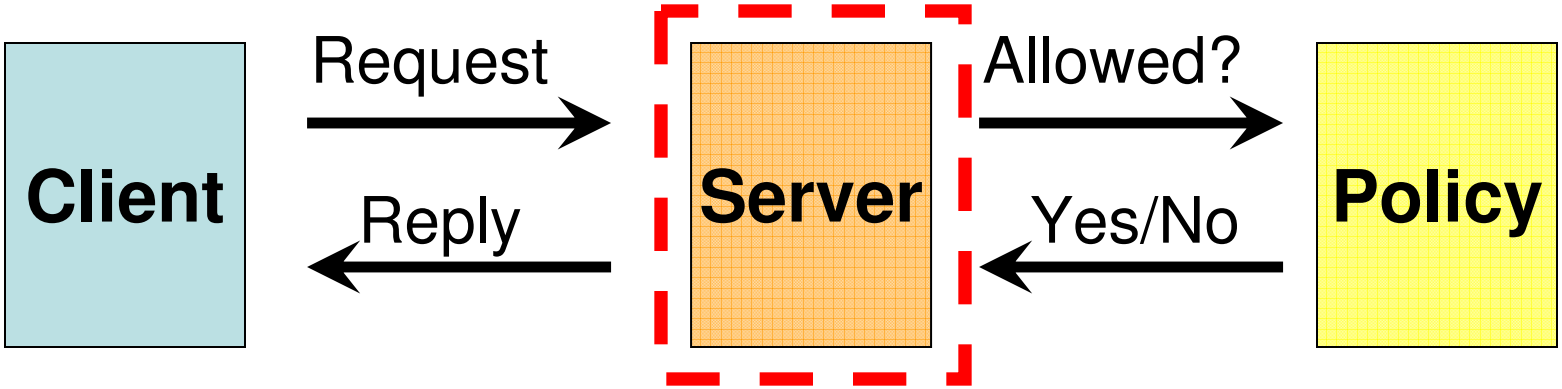
Many more examples

- Prevent unauthorized
  - copy-and-paste **[Epstein *et al.,* 1991]**
  - modification of inputs meant for other clients
  - changing window settings of other clients
  - retrieval of bitmaps: screenshots
  - …several more examples…

Source: **[Kilpatrick *et al.,* 2003]**

Fine-grained enforcement

- Fine-grained, server-level enforcement of authorization policies



- X Client → X Server: Give me input keystrokes
- X Server → Policy Engine: Is this allowed?
- X Server → X Client: Here are the keystrokes

Problem statement

- Provide server-level mechanisms for enforcement of authorization policies
- Make server code security-policy-aware

Contributions

- Analyses for legacy code retrofits
  - Enforcing authorization policies
- Fingerprints
  - Code-patterns of security-sensitive operations
- Two prototype tools
  - AID: automates fingerprint-finding
  - ARM: uses fingerprints to retrofit code
- Real-world case study
  - Retrofitting the X server

Talk outline

- Motivation and contributions
- Retrofitting legacy code: Lifecycle
- Our techniques
  - Fingerprints
  - Finding fingerprints: AID
  - Using fingerprints: ARM
- Conclusion

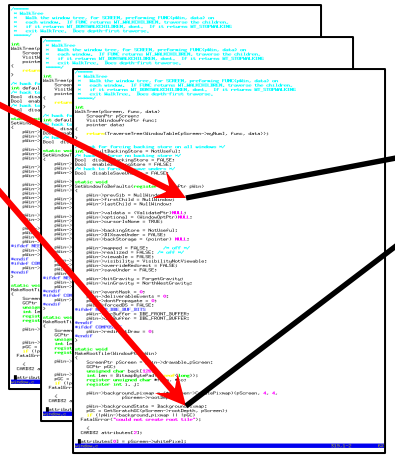
Retrofitting legacy code: Lifecycle

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

Security-sensitive operations

INPUT_EVENT
CREATE
DESTROY
COPY
PASTE
MAP
. . .

Source Code



Policy checks

Can the client receive this **INPUT_EVENT**?

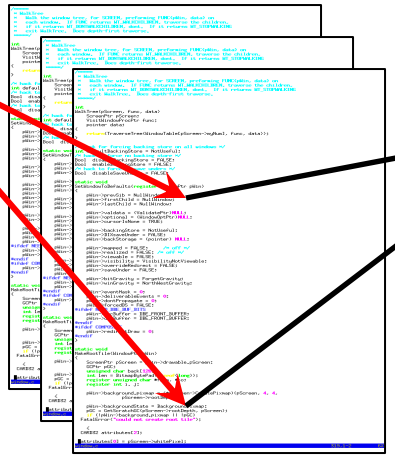
Lifecycle: State-of-the-art

Security-sensitive operations

INPUT_EVENT
CREATE
DESTROY
COPY
PASTE
MAP
. . .

Source Code

Policy checks



Can the client receive this **INPUT_EVENT**?

State-of-the-art: Consequences

- Tedious
  - Linux Security Modules ~ 2 years **[Wright *et al.,* 2002]**
  - X11/SELinux ~ 2 years **[Kilpatrick *et al.,* 2003]**

- Error-prone
  - Violation of complete mediation **[Jaeger et al. 2002]**

Talk outline

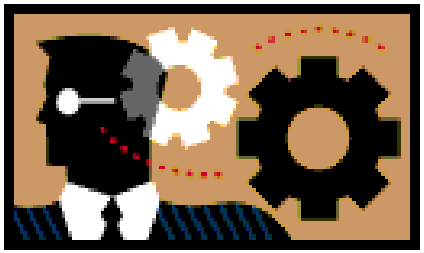
- Motivation and contributions
- Retrofitting legacy code: Lifecycle
- Our techniques
  - Fingerprints
  - Finding fingerprints: AID
  - Using fingerprints: ARM
- Conclusion

Lifecycle: Our contributions

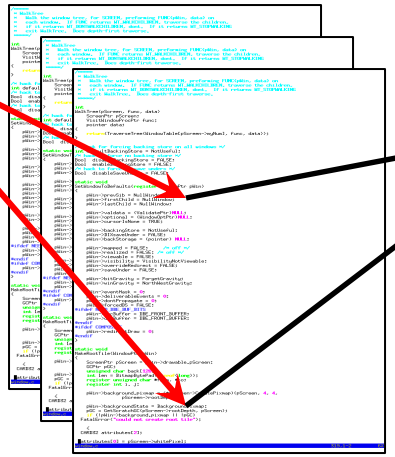
Security-sensitive operations

- `INPUT_EVENT`
- `CREATE`
- `DESTROY`
- `COPY`
- `PASTE`
- `MAP`
- . . .

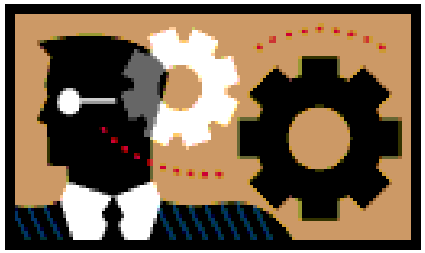
AID



Source Code



ARM



Policy checks

Can the client receive this `INPUT_EVENT`?

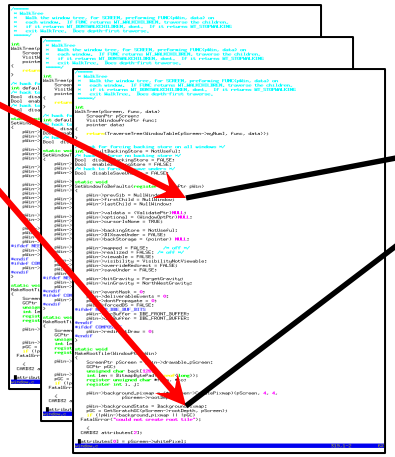
Overview of our work

- Operations on shared resources
- Manually identified list
  - For X server, used NSA study [Kilpatrick *et al.*, 2003]

Security-sensitive operations

- **INPUT_EVENT**
- **CREATE**
- **DESTROY**
- **COPY**
- **PASTE**
- **MAP**
- . . .

Source Code



Policy checks

Can the client receive this **INPUT_EVENT**?

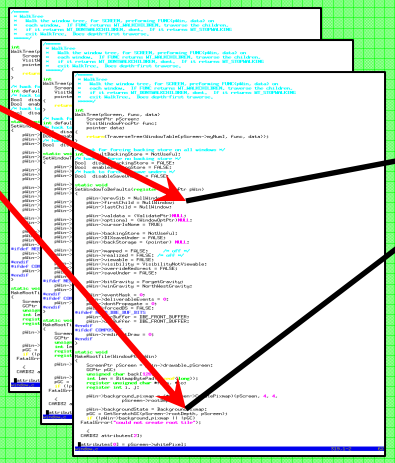
Overview of our work

- Main concept: fingerprints
- Approach: analysis of runtime traces

Security-sensitive operations

INPUT_EVENT
CREATE
DESTROY
COPY
PASTE
MAP
. . .

Source Code



Policy checks

Can the client receive this **INPUT_EVENT**?

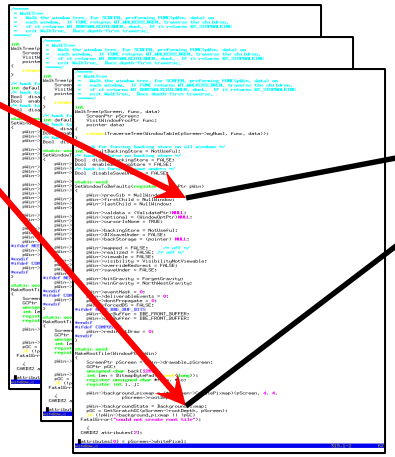
Overview of our work

- Main concept: reference monitoring
- Approach: static matching of fingerprints
  [Ganapathy/Jaeger/Jha, CCS'05]

Security-sensitive operations

INPUT_EVENT
CREATE
DESTROY
COPY
PASTE
MAP
. . .

Source Code



Policy checks

Can the client receive this **INPUT_EVENT**?

Talk outline

- Motivation
- Case study: X window system
- Retrofitting legacy code: Lifecycle
- Our techniques
  - Fingerprints
  - Finding fingerprints: AID
  - Using fingerprints: ARM
- Conclusion

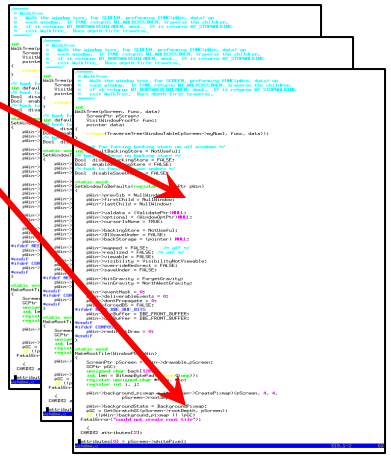
What are fingerprints?

- Code-level description of security-sensitive operations
- Each operation has at least one fingerprint

Security-sensitive operations

```
INPUT_EVENT
CREATE
DESTROY
COPY
PASTE
MAP
...
```

Source Code



Examples of Fingerprints

Code-patterns

- **INPUT_EVENT :-**

  Call `ProcessKeybdEvent`

- **INPUT_EVENT :-**

  Call `ProcessPointerEvent`

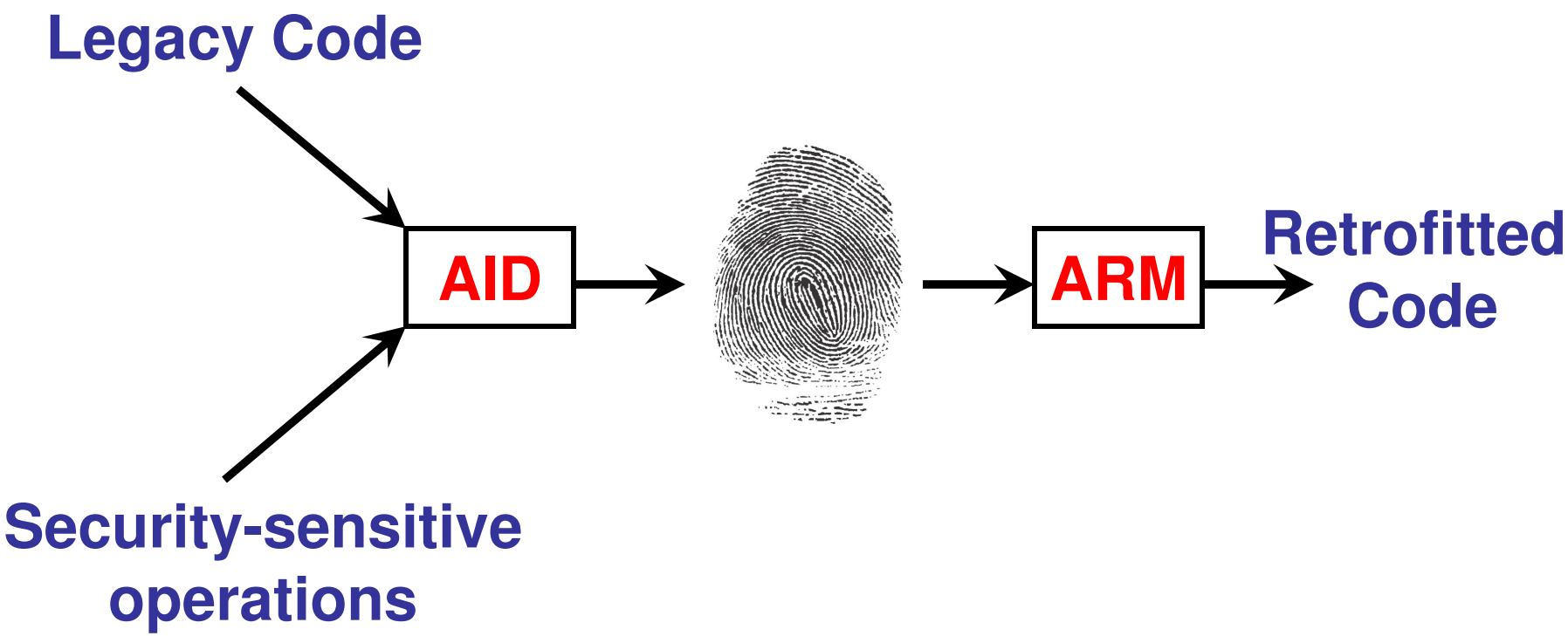
- **ENUMERATE:-**

  Read `Window->firstChild &`

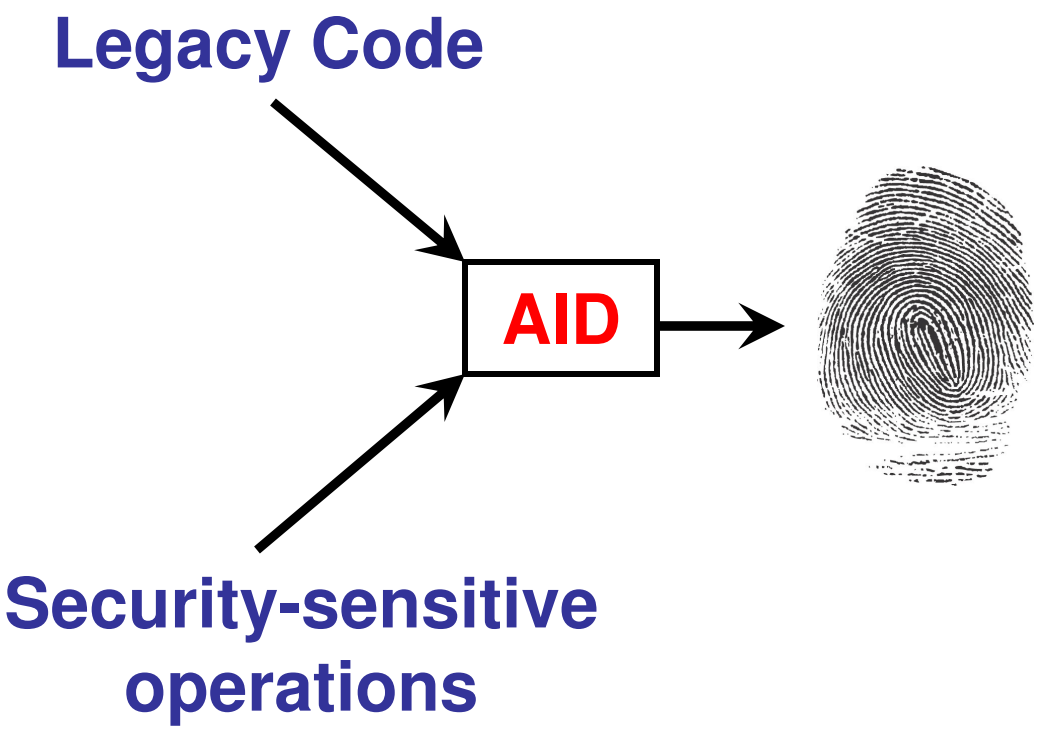
  Read `Window->nextSib &`

  Compare `Window ≠ 0`

Finding and using fingerprints



AID: A fingerprint finder



Main problem solved by AID

- **Inputs:**
  1. Source code of legacy server
  2. Security-sensitive operations
- **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

- **Output:** Fingerprints

Key insight used by AID

- Induce server to perform a security-sensitive operation
  - typing to window will induce `INPUT_EVENT`
- Code-patterns in its fingerprint must be exercised by the server
  - ***Call*** `ProcessKeybdEvent` must be in trace
- Analyze runtime traces to find fingerprints!

Runtime traces

- Trace the server and record
  - function calls and returns
  - reads/writes to critical data structures
    - Data structures used to represent resources
- Example: from X server startup

```
CALL SetWindowToDefaults
 SET Window->prevSib TO 0
 SET Window->firstChild TO 0
 SET Window->lastChild TO 0
 … about 1400 such code-patterns
```

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`**

Analyzing traces

- **Input:**
  - Traces annotated with the security-sensitive operations they perform
- **Output:**
  - Fingerprint for each security-sensitive operation

Analyzing traces: “diff” and “∩”

Annotation is currently a manual step

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

Analyzing traces: “diff” and “∩”

Perform same set operations on code-patterns in traces

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

CREATE = Trace1 ∩ Trace4 - Trace 3

How effective is trace analysis?

- Source code: **1,000,000** lines of C code
- Raw traces: **54,000** code-patterns
- Pre-analysis: Relevant portion of trace
  - Average of **900** *distinct* code-patterns
  - Average of **140** *distinct* functions
- Post-analysis: Each result
  - Average of **126** *distinct* code-patterns
  - Average of **15** *distinct* functions

18x

~60x

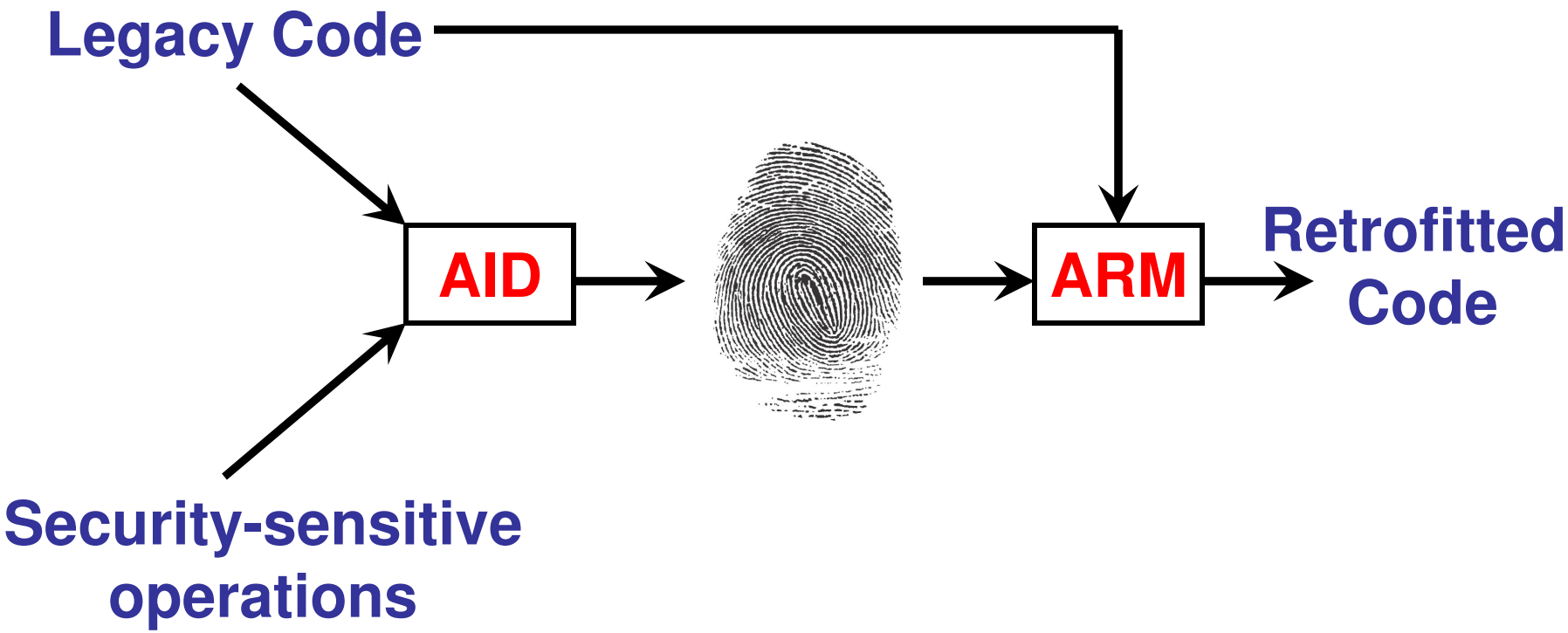
~7x

~9x

Examples of fingerprints

Operation	Fingerprint
CREATE	***Call*** `CreateWindow`
DESTROY	***Call*** `DeleteWindow`
UNMAP	***Set*** `xEvent->type` ***To*** `UnmapNotify`
CHSTACK	***Call*** `MoveWindowInStack`
INPUT_EVENT	***Call*** `ProcessPointerEvent`, ***Call*** `ProcessKeybdEvent`

ARM: Static code retrofitter



Fingerprints from AID

Operation	Fingerprint
CREATE	***Call*** `CreateWindow`
DESTROY	***Call*** `DeleteWindow`
UNMAP	***Set*** `xEvent->type` ***To*** `UnmapNotify`
CHSTACK	***Call*** `MoveWindowInStack`
INPUT_EVENT	***Call*** `ProcessPointerEvent`, ***Call*** `ProcessKeybdEvent`

Using fingerprints: simple example

```
CreateWindow(Client *pClient) {
    Window *pWin;
    …
    // Create new window here
    pWin = newly-created window;
}
```

```
CreateWindow(Client *pClient) {
    Window *pWin;
    if (CHECK(pClient, CREATE) == FAIL) { return; }
    // Create new window here
    pWin = newly-created window;
}
```

More complex example

- **`ENUMERATE :-`**

  Read `Window->firstChild` **&**

  Read `Window->nextSib` **&**

  Compare `Window ≠ 0`

- Paper has details on how we match these

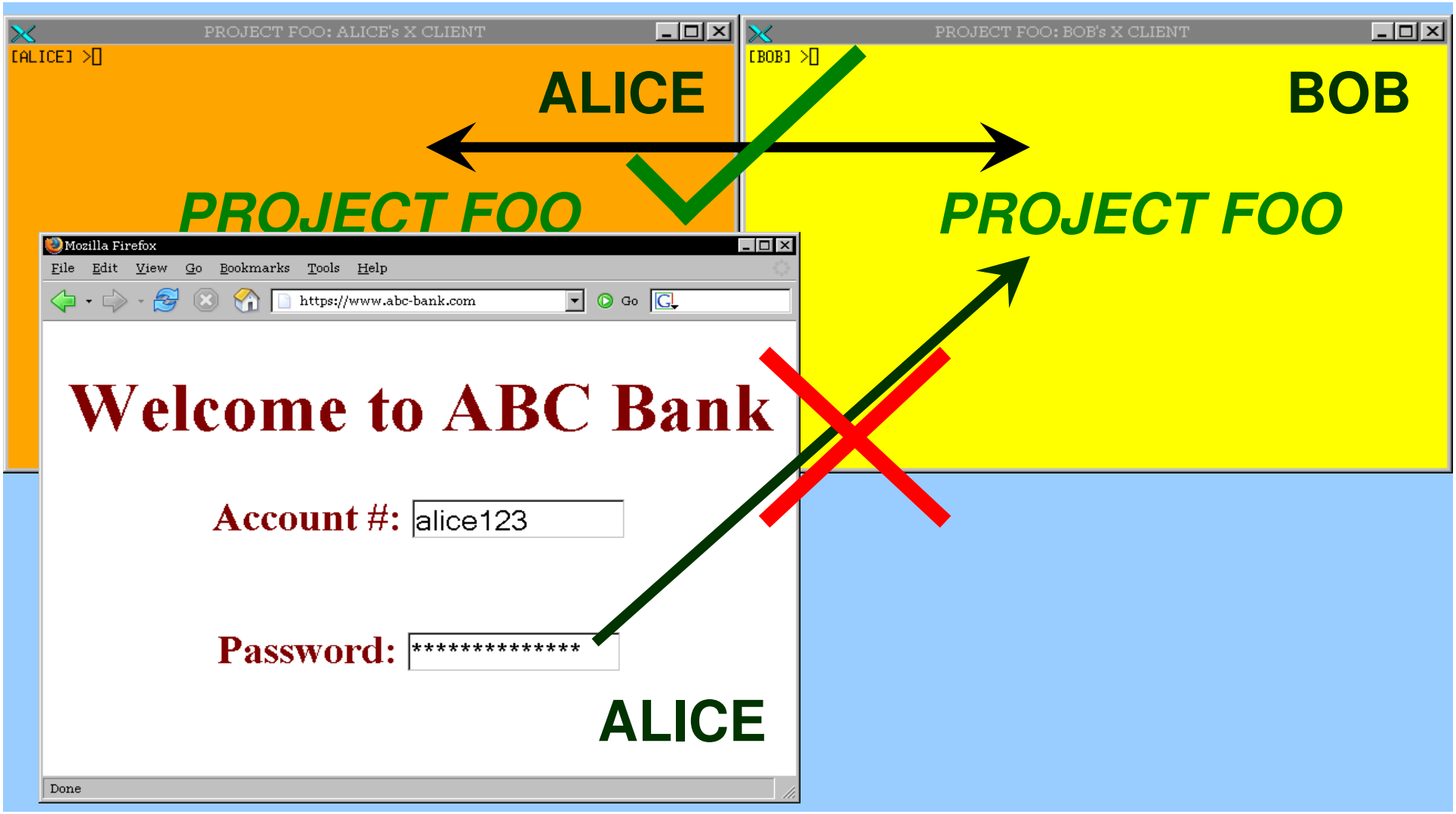
Talk outline

- Motivation
- Case study: X window system
- Retrofitting legacy code: Lifecycle
- Our techniques
  - Fingerprints
  - Finding fingerprints: AID
  - Using fingerprints: ARM
- Conclusion

X server case study

- Applied AID and ARM to the X server
- Added policy checks for window operations
  - Policy lookups at 24 locations

Similar example in the paper



Limitations

1. AID uses analysis of runtime traces
   - no guarantees of finding all fingerprints
   - Possible remedies
     - coverage metrics to augment runtime tracing
     - static fingerprint-finding technique

2. Identification of security-sensitive operations is still manual

Summary of important ideas

- Analysis techniques to retrofit servers for policy enforcement
- Fingerprints
  - Code-patterns of security-sensitive operations
- Two prototype tools
  - AID: automates fingerprint-finding
  - ARM: uses fingerprints to retrofit code
- Case study on X server

Questions?

Retrofitting Legacy Code for Authorization Policy Enforcement

Vinod Ganapathy
vg@cs.wisc.edu

Trent Jaeger
tjaeger@cse.psu.edu

Somesh Jha
jha@cs.wisc.edu

http://www.cs.wisc.edu/~vg/papers/ieee-sp2006