Retrofitting Legacy Code for Security

Vinod Ganapathy
Computer Sciences Department
University of Wisconsin-Madison
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

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

Resource manager

Allowed?

Reference monitor

Response

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]
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
- Future work
X server with multiple X clients
Malicious remote X client

Welcome to ABC Bank

Account #: alice123

Password: ************
Undesirable information flow

Welcome to ABC Bank

Account #: alice123
Password: ************
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

X server

Reference monitor

Allowed?

Authorization policy

Response

YES/NO
Outline

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

Legacy code

Miner

Fingerprints

Matcher

Retrofitted code
Outline

- Motivation
- Problem

- Solution
  - Fingerprints
  - Dynamic fingerprint mining
  - Static fingerprint mining

- Future work
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

- \( \text{Input\_Event} :\) 
  \[ \text{Cmp} \ x\text{Event->type == KeyPress} \]

- \( \text{Input\_Event} :\) 
  \[ \text{Cmp} \ x\text{Event->type == MouseMove} \]

- \( \text{Map} :\) 
  \[ \text{Set} \ Window->mapped \text{ to True} \ & \]
  \[ \text{Set} \ x\text{Event->type to MapNotify} \]

- \( \text{Enumerate} :\) 
  \[ \text{Read} \ Window->\text{firstChild} \ & \]
  \[ \text{Read} \ Window->nextSib \ & \]
  \[ \text{Cmp} \ Window \neq 0 \]
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**

```c
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
- Future work
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]

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input_Event</strong></td>
<td>Input to window from device</td>
</tr>
<tr>
<td><strong>Create</strong></td>
<td>Create new window</td>
</tr>
<tr>
<td><strong>Destroy</strong></td>
<td>Destroy existing window</td>
</tr>
<tr>
<td><strong>Map</strong></td>
<td>Map window to console</td>
</tr>
</tbody>
</table>

- 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 \( s \in S \) 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 \textit{Input\_Event}
  - Creating new window generates \textit{Create}
  - Creating window also generates \textit{Map}
  - Closing existing window generates \textit{Destroy}
Comparison with “diff” and “∩”

<table>
<thead>
<tr>
<th></th>
<th>Open xterm</th>
<th>Close xterm</th>
<th>Move xterm</th>
<th>Open browser</th>
<th>Switch windows</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Create</strong></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Destroy</strong></td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Map</strong></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Unmap</strong></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Input_Event</strong></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Annotation is a manual step
## Comparison with “diff” and “∩”

Perform same set operations on resource accesses

<table>
<thead>
<tr>
<th></th>
<th>Open xterm</th>
<th>Close xterm</th>
<th>Move xterm</th>
<th>Open browser</th>
<th>Switch windows</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Create</strong></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Destroy</strong></td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Map</strong></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Unmap</strong></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Input_Event</strong></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

**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** \(\cap\) Open browser – Move **xterm**

- Perform the same set operations on the set of resource accesses in each trace
Experimental methodology

1. **Source code**
   - gcc --enable-logging
2. **Server with logging enabled**
   - Run experiments and collect traces
3. **Raw traces**
   - Localize security-sensitive operation
4. ** Relevant portions of traces**
   - Compare traces with “diff” and “∩”
5. **Pruned traces**
Dynamic mining: Results

Each fingerprint localized to within 126 resource accesses

Size

Source Code | Raw Traces | Relevant Portions | Pruned Traces

1,000,000 | 54,000 | | 126
Limitations of dynamic mining

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

- Future work
Static fingerprint mining

Security-sensitive operations

- Input_Event
- Create
- Destroy
- Copy
- Paste
- Map

Source Code

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 \( P = \{C_1, C_2, \ldots, C_n\} \) of **R**
  - \( R = C_1 \cup C_2 \cup \ldots \cup C_n \)
  - Resource accesses in each trace of the program are composed of elements of **P**
Problem definition

- C₁, C₂, ..., Cₙ 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: \texttt{MapSubWindows(...)}
  - Resource accesses:
    - \texttt{Set xEvent->type To MapNotify}
    - \texttt{Set Window->mapped To True}
    - \texttt{Read Window->firstChild}
    - \texttt{Read Window->nextSib}
    - \texttt{Cmp Window \neq 0}
## Resource accesses

<table>
<thead>
<tr>
<th></th>
<th>MapSub Windows</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Set xEvent-&gt;type To MapNotify</td>
<td>✔️ ✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Identify candidate fingerprints by comparing resource accesses</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read Window-&gt;nextSib</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmp Window ≠ 0</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**270 API functions**

**430 distinct resource accesses**

Identify candidate fingerprints by comparing resource accesses.
## Concept analysis

### Instances

<table>
<thead>
<tr>
<th>Set xEvent-&gt;type To MapNotify</th>
<th>MapSub Windows</th>
<th>Map Window</th>
<th>Keyboard Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Window=</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read Window</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read Window-&gt;nextSib</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmp Window ≠ 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmp xEvent-&gt;type==KeyPress</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Comparison via hierarchical clustering**
Clustering via concept analysis

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set xEvent-&gt;type To MapNotify</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>Set Window-&gt;mapped To True</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>Read Window-&gt;firstChild</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>Read Window-&gt;nextSib</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>Cmp Window ≠ 0</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>6</td>
<td>Cmp xEvent-&gt;type==KeyPress</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\{A,B,C\}, \Phi \\
\{A,B\}, \{1,2\} \\
\{A\}, \{1,2,3,4,5\} \\
\Phi, \{1,2,3,4,5,6\} \\
\{C\}, \{6\}
\end{align*}
\]
Mining candidate fingerprints

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set xEvent-&gt;type To MapNotify</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>Set Window-&gt;mapped To True</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>Read Window-&gt;firstChild</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Read Window-&gt;nextSib</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Cmp Window ≠ 0</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>6</td>
<td>Cmp xEvent-&gt;type==KeyPress</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Cand. Fing. 1

Cand. Fing. 2

Cand. Fing. 3

\{A,B,C\}, \Phi

\{A,B\}, \{1,2\}

\{A\}, \{1,2,3,4,5\}

\Phi, \{1,2,3,4,5,6\}

\{C\}, \{6\}
## Static mining: Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>LOC</th>
<th>Cand. Fing.</th>
<th>Avg. Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext2</td>
<td>4,476</td>
<td>18</td>
<td>3.7</td>
</tr>
<tr>
<td>X Server/dix</td>
<td>30,096</td>
<td>115</td>
<td>3.7</td>
</tr>
<tr>
<td>PennMUSH</td>
<td>94,014</td>
<td>38</td>
<td>1.4</td>
</tr>
</tbody>
</table>

![Bar chart showing size comparison for ext2, X server, and PennMUSH]
### Static mining: Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Manually identified Security-sensitive ops</th>
<th>Candidate fingerprints</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext2</td>
<td>11</td>
<td>18</td>
</tr>
<tr>
<td>X Server/dix</td>
<td>22</td>
<td>115</td>
</tr>
</tbody>
</table>

Able to find **at least one fingerprint** for each security-sensitive operation.
## Static mining: Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Manually identified</th>
<th>Candidate fingerprints</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext2</td>
<td>11</td>
<td>18</td>
</tr>
<tr>
<td>X Server/dix</td>
<td>22</td>
<td>115</td>
</tr>
</tbody>
</table>

Identified as part of multi-year efforts in ν minutes, κ hours.
## Static mining: Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Manually identified security-sensitive ops</th>
<th>Candidate fingerprints</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext2</td>
<td>11</td>
<td>18</td>
</tr>
<tr>
<td>X Server/dix</td>
<td>22</td>
<td>115</td>
</tr>
</tbody>
</table>

- 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]

Input Event
Create
Destroy
Copy
Paste
Map

Can the client receive this Input_Event?
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
Outline

- Motivation
- Problem
- Solution
- Future work
Short term plans

- 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
Long term: Better containment

- Several software systems are monolithic
- Poor containment: Attacks compromise the entire system
Refactoring for containment

- Detect and isolate compromised components
- Transparently recover compromised components
- Minimize communication overhead
Long term: Formal study of policy

- SELinux example policy: 50,000 statements
  - Security guarantees: ?
  - Security/application functionality tradeoff: ?
  - Maintenance and upgrade: ?

Can the client receive this Input_Event?
Other research

- Static buffer overrun detection [CCS’03]
- API-level exploit discovery [ICSE’05]
- Detecting heap corruptions [ASPLOS’06]
- Spyware signature generation [ACSAC’06]
- Fraud detection on eBay [CCS’05]
- Slicing synchronous programs [ENTCS’02]
- Refactoring device drivers [HotOS’07]
Acknowledgements

- Trishul Chilimbi
- Mihai Christodorescu
- Jonathon Giffin
- Trent Jaeger
- Somesh Jha
- David King

- Louis Kruger
- Thomas Reps
- Shai Rubin
- Sanjit Seshia
- Michael Swift
- Hao Wang
Retrofitting Legacy Code for Security

Vinod Ganapathy
Computer Sciences Department
University of Wisconsin-Madison
vg@cs.wisc.edu
http://www.cs.wisc.edu/~vg
BACKUP SLIDES
Other research

- Static buffer overrun detection  [CCS 2003]
  - Addressed context-sensitive buffer overrun detection using linear programming

- API-level exploit discovery  [ICSE 2005]
  - Designed formal framework to find exploitable vulnerabilities

- Detecting heap corruptions  [ASPLOS 2006]
  - Found bugs via anomaly detection on heap-graph metrics
Other research

- Signature generation \[\text{[ACSAC 2006]}\]
  - Characterized network-level behavior of spyware

- Fraud detection on eBay \[\text{[CCS 2005]}\]
  - Detected price inflation via statistical models of seller behavior

- Refactoring device drivers \[\text{[HotOS 2007]}\]
  - Automatically refactored device drivers into \textit{microdrivers} to improve fault isolation