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

- 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

Most deployed software is not designed for security

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

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]

Enforcing authorization policies

Resource manager
Enforcing authorization policies
Resource user
Operation request Response
Authorization policy
Allowed?
<Alice, /etc/passwd, File_Read>

Contributions

- 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

Analyses and transformations for authorization policy enforcement
Outline

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

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]

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

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
  - Fingerprint
  - Matcher
  - Retrofitted code
Outline

- Motivation
- Problem
- Solution
  - Fingerprints [CCS’05]
    - Dynamic fingerprint mining
    - Static fingerprint mining
- Future work

Outline

- Motivation
- Problem
- Solution
  - Fingerprints [CCS’05]
    - Dynamic fingerprint mining
    - Static fingerprint mining
- Future work

What are fingerprints?

- 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

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

- 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

Placing authorization checks

- X server function `MapSubWindows`

Refer to the code snippet for implementation details.

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: Fingersprints

```
Input_Event :-
Cmp xEvent->type == KeyPress
```

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 ∈ 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**

Dynamic fingerprint mining

- **Security-sensitive operations**
  
<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input_Event</td>
<td>Input to window from device</td>
</tr>
<tr>
<td>Create</td>
<td>Create new window</td>
</tr>
<tr>
<td>Destroy</td>
<td>Destroy existing window</td>
</tr>
<tr>
<td>Map</td>
<td>Map window to console</td>
</tr>
</tbody>
</table>

[NSA'03]

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

Traces contain fingerprints

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

<table>
<thead>
<tr>
<th>Annotation is a manual step</th>
</tr>
</thead>
</table>

<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>Create</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Destroy</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Map</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Unmap</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Input_Event</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

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
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>Create</td>
<td>✓</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destroy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Map</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unmap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input_Event</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 ∩ Open browser – Move xterm

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

Experimental methodology

1. Source code
   \[\text{gcc --enable-logging}\]
2. Server with logging enabled
3. Raw traces
4. Relevant portions of traces
5. Pruned traces

Dynamic mining: Results

- Each fingerprint localized to within 126 resource accesses
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

Runtime trace

Output: Candidate Fingerprints

Cmp xEvent->type == KeyPress

Resources
- Window
- xEvent

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 = {C1, C2, …, Cn} of R
  - R = C1 U C2 U … U Cn
  - Resource accesses in each trace of the program are composed of elements of P
Problem definition

- \( C_1, C_2, \ldots, C_n \) 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: \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 ≠ 0}

Resource accesses

Identify candidate fingerprints by comparing resource accesses

<table>
<thead>
<tr>
<th></th>
<th>MapSub</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{Set xEvent-&gt;type To MapNotify}</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>\texttt{Set Window-&gt;mapped To True}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\texttt{Read Window-&gt;firstChild}</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>\texttt{Read Window-&gt;nextSib}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\texttt{Cmp Window ≠ 0}</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Keyboard Input

Map Window

MapSub Windows

270 API functions

430 distinct resource accesses

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

<table>
<thead>
<tr>
<th>Instances</th>
<th>MapSub Windows</th>
<th>Map Window</th>
<th>Keyboard Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set xEvent-&gt;type To MapNotify</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<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>

\{A,B,C\}, \{C\}, \{6\}

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

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

\{C\}, \{6\}

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

### Mining candidate fingerprints

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

\{A,B,C\}, \{C\}, \{6\}

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

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

\{C\}, \{6\}

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

### 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>

![Graph showing size distribution 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

---

Summary of contributions

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

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

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  [ACSAC 2006]
  - Characterized network-level behavior of spyware
- Fraud detection on eBay  [CCS 2005]
  - Detected price inflation via statistical models of seller behavior
- Refactoring device drivers  [HotOS 2007]
  - Automatically refactored device drivers into microdrivers to improve fault isolation