Advanced Systems Security: Program Retrofitting

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University
Problem

- Write a program to provide a valuable function
  - Process any users’ data
  - Some data may be security critical
- What could go wrong?
Problem: Input Filtering

- Programs may receive maliciously crafted inputs
  - To take control of the program
  - To change control of program
  - To control names produced by program

- Programs need to filter inputs to prevent a variety of attacks

- Adversaries may use your program to implement their goals
Problem: Bypass

- Programs may receive requests from multiple parties (clients)
  - Need to authenticate source of request
  - Need to authorize access to program resources
  - Either may be missing
    - Leading to unauthorized access to resources

- Adversaries may steal or illicitly modify critical program data
Problem: System Data Attack

• Programs may process security-critical system resources
  ‣ Adversaries may be able to induce program into leaking such resources
  ‣ Or maliciously modifying them for adversaries
  ‣ And programs may run on many systems…

• How can programs be deployed on many systems and enforce necessary security requirements?
History of Secure Programming

- Programmers ignore security
- If programmers just knew about security problems they would fix them
- Stop using insecure C library functions
- System call APIs are extended to help programmers address problems
- Some languages add filtering functions
- ...
Complex Program Checks

- Check for symbolic link (lstat)
- Check for lstat-open race
- Check for inode recycling
- Do checks for each path component (safe_open)
  - /, var, mail, …
- Cannot expect programmers to get this right!

```c
/* fail if file is a symbolic link */
int open_no_symlink(char *fname)
{
    struct stat lbuf, buf;
    int fd = 0;
    lstat(fname, &lbuf);
    if (S_ISLNK(lbuf.st_mode))
        error("File is a symbolic link!");
    fd = open(fname);
    fstat(fd, &buf);
    if (((buf.st_dev != lbuf.st_dev) ||
        (buf.st_ino != lbuf.st_ino))
        error("Race detected!");
    lstat(fname, &lbuf);
    if (((buf.st_dev != lbuf.st_dev) ||
        (buf.st_ino != lbuf.st_ino))
        error("Cryogenic sleep race!");
    return fd;
}
```
Inefficient Program Checks

- Checking retrieved resources is expensive
  - Single open() requires $4 \times \text{path length}$ additional syscalls
  - Programmers omit checks to improve performance
- Example: Apache documentation recommends switching off resource access checks

```xml
<Directory /www/htdocs>
  Options SymLinksIfOwnerMatch
</Directory>
```

And a request is made for the URI `/index.html`. Then Apache will perform `lstat(2)` on `/www/htdocs` and `/www/htdocs/index.html`. The results of these `lstat`es are never cached, so they will occur on every single request. If you really desire the symlink security checking you can do something like this:

```xml
<Directory /www/htdocs>
  Options FollowSymLinks
</Directory>

<Directory /www/htdocs>
  Options -FollowSymLinks +SymLinksIfOwnerMatch
</Directory>
```

This at least avoids the extra checks for the `DocumentRoot` path. Note that you'll need to add similar sections if you have any `Alias` or `RewriteRule` paths outside of your document root. For highest performance, and no symlink protection, set `FollowSymLinks` everywhere, and never set `SymLinksIfOwnerMatch`. 
Goal: Automate Security

• Given a program
  ‣ And a security goal

• Produce a retrofitted program
  ‣ That enforces that goal
Examples

- StackGuard
  - Add canary for you

- Binary Rewriting
  - For Software Fault Isolation
  - And CFI

- Uh, not many more – at least in practice
Authorizing Access

- **Resource user**
- **Operation request**
- **Resource manager**
- Reference monitor
- **Allowed?**
- **YES/NO**
- `<Alice, /etc/passwd, File_Read>`
Authorizing Access

- **Resource user**
  - Operation request
  - Resource manager
    - Authorization Hooks
    - Reference monitor
      - Allowed?
        - YES/NO
        - Authorization policy
          - Response
Retrofitting Legacy Code

- What if you had to place authorization hooks to add a reference monitor into a legacy program?

Need systematic techniques to retrofit legacy code for security

Legacy code \[\rightarrow\] Retrofitted code

INSECURE \[\rightarrow\] SECURE
Retrofitting for Authorization

• Mandatory access control for Linux
  ‣ Linux Security Modules [Wright et al.,’02]
  ‣ TrustedBSD, SEDarwin, sHype, XSM, …

• Secure windowing systems
  ‣ 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]
• And more: Apache, PostgreSQL, dbus, gconf, …
Retrofitting for Authorization

- X11 ~ proposed 2003, upstreamed 2007, changing to date. [Kilpatrick et al., ‘03]

- Linux Security Modules ~ 2 years [Wright et al., ’02]

- PostgreSQL: Began in 2006, still not mainline

Painstaking, manual procedure

At this point, SE-PostgreSQL has taken up a *lot* of community resources, not to mention an enormous and doubtless frustrating amount of *the lead developer’s* time and effort, thus far without a single committed patch, or even a consensus as to what it should (or could) do. Rather than continuing to blunder into the future, I think we need to do a reality check

- http://archives.postgresql.org/message-id/20090718160600.GE5172@fetter.org
Past Efforts

- **Automated Hook Verification:**
  - **Assumptions: Training wheels**
    - (sensitive data types, existing hook placement)
    - For Kernels
      - [Zhang et al., 2002, Edwards et al., 2002, Tan et al., 2008]
- **Automated Hook Placement:**
  - **Assumptions: Training wheels**
    - (sensitive data types, hook code)
    - For Server Applications
      - [Ganapathy et al., 2005, 2006, 2007]
    - Web Applications
      - [Sun et al., 2011, RoleCast 2011]
Goal Statement

We want to generate authorization hook placements mostly-automatically for legacy code with minimal programmer effort.
Outline

• Motivation

• Problem
  ‣ Example
  ‣ Retrofitting legacy code: Lifecycle

• Solution
  ‣ Static mining from “user choice”
  ‣ Understanding placement options using “dominance”
X Server & Many X Clients

Welcome to ABC Bank

Account #: alice123

Password: **************

LOCAL

REMOTE
Malicious Remote X Client

Welcome to ABC Bank

Account #: alice123

Password: ***************

LOCAL
Illegal Information Flow

Welcome to ABC Bank

Account #: alice123

Password: **************

REMOTE

LOCAL
Desirable Information Flow
Hook Placement: Objects

Program

Challenges

- Identifying Objects
Hook Placement: Objects

Program

Challenges

- Identifying Objects:
  - Programs manipulate many variables
    - 7800 in X
    - Of over 400 data types
Hook Placement: Objects

Program

Challenges

• Identifying Objects:
  ‣ Programs manipulate many variables
  ‣ Only a subset are security-sensitive
    • Available to all?
    • Provably unavailable (modulo type-safety)
Hook Placement: Operations

Program

Challenges

- Identifying Operations:
  - Set of instructions
Hook Placement: Operations

Program

Challenges

- Identifying Operations:
  - Set of instructions
  - Only a subset are security-sensitive
Hook Placement Problem

Program

Challenges

- Place Authorization Hooks.
- Must meet reference monitor guarantees.
  - Completeness
  - Verifiability
  - Tamper-proof
Hook Placement Problem

Program

Challenges

• What is ideal placement?
  › Minimal number of hooks?
• May block legitimate function
Hook Placement Problem

Program

Challenges

- What is ideal placement?
  - Minimize redundancy?
  - What if \{CD\} and \{KL\} perform same security-sensitive operation on same security-sensitive objects?
  - No need for second hook.
Outline

• Motivation

• Problem
  ▸ Example
  ▸ Retrofitting legacy code: Lifecycle

• Solution
  ▸ Static mining from “user choice”
  ▸ Understanding placement options using “dominance”
Idea: Request Choices

• In servers, *user-request* determines *choices* that client subjects can make in the program.

• “Choice”:
  ‣ Determines which *objects* are selected to be operated upon.
  ‣ Determines which *operation* is selected to be performed on objects.
Inputs and Containers
Using Taints for Lookup

Program

User A
User B

Request Interface \( i \)

\( v = \text{Lookup}(O,i) \)

Predicated on tainted variable

Container

\( O \)

A

B

C

D

E

F

I

H

J

K

L
Security-Sensitive Object

Request Interface \( i \)

Program

\( A \) → \( B \)

\( C \)

\( v = \text{Lookup}(O, i) \)

Op 1.0

\( D \) → \( E \)

\( F \) → \( I \) → \( K \)

\( H \) → \( J \) → \( L \)

Container \( O \)

\( o1 \) → \( o2 \) → \( o3 \) → \( o4 \)
Tainted Predicate

Control statement predicated on tainted variable
User-Choice Operations

\[ v = \text{Lookup}(O, i) \]

User choice operation

Program

Container \( O \)

User A

User B

Request Interface \( i \)

Op 1.0

Op 1.1

Op 1.3

Op 1.3

read \( v \)

write \( v \)
Security-Sensitive Operations

Program

A

Request Interface i

B

C

\( v = \text{Lookup}(O,i) \)

D

E

Op 1.0

F

H

read v

Op 1.1

I

J

Op 1.3

K

L

write v

Op 1.3

User A

User B

Container O

Security-sensitive operation
Hook Placement Method

Stage 1
Identify tainted variables

Stage 2
Identify security-sensitive objects

Stage 3
Identify user-choice operations

Stage 4
Identify authorization hook placements

Source code

User request

Hook Placement Tool

Lookup function specification
Global read specification
Optimization Requirements

**Goals:**
1. Hoist authorization hooks as early as possible
2. Eliminate redundant hook placements

---

**Stage 1**
Identify tainted variables

**Stage 2**
Identify security-sensitive objects

**Stage 3**
Identify user-choice operations

**Stage 4**
Identify authorization hook placements

- Identify security-sensitive operations
- Identify optimized hook placements

---

**Hook Placement Tool**

---

**Lookup function specification**

**Global read Specification**
Hoisting

Structure member access to mediate

Connection from outside this procedure

Operation

control statement

Dummy node
Removal

Connection from outside this procedure

Structure member access to mediate

Control statement

Dummy node

Op 1.1
Op 2.1
Op 2.2
Placement
## Results

<table>
<thead>
<tr>
<th>Program</th>
<th>X Server</th>
<th>postgres</th>
<th>pennmush</th>
<th>memcached</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>28k</td>
<td>49k</td>
<td>78k</td>
<td>9k</td>
</tr>
<tr>
<td><strong>Total variables</strong></td>
<td>7795</td>
<td>12350</td>
<td>24372</td>
<td>2350</td>
</tr>
<tr>
<td>Tainted variables</td>
<td>2975 (38%)</td>
<td>5100 (41%)</td>
<td>4168 (17%)</td>
<td>490 (20%)</td>
</tr>
<tr>
<td>Security sensitive variables</td>
<td>823 (10%)</td>
<td>402 (3%)</td>
<td>1573 (6%)</td>
<td>82 (3%)</td>
</tr>
<tr>
<td><strong>Data Structures</strong></td>
<td>404</td>
<td>278</td>
<td>311</td>
<td>41</td>
</tr>
<tr>
<td>Sensitive Data structures</td>
<td>61 (15%)</td>
<td>30 (10%)</td>
<td>38 (12%)</td>
<td>7 (17%)</td>
</tr>
<tr>
<td><strong>User-choice Operations</strong></td>
<td>4760</td>
<td>5063</td>
<td>6485</td>
<td>996</td>
</tr>
<tr>
<td>Sensitive operations</td>
<td>1382 (29%)</td>
<td>1378 (27%)</td>
<td>1382 (21%)</td>
<td>203 (20%)</td>
</tr>
<tr>
<td>Hooks</td>
<td>532 (11%)</td>
<td>579 (11%)</td>
<td>714 (11%)</td>
<td>56 (5%)</td>
</tr>
</tbody>
</table>
# Results

<table>
<thead>
<tr>
<th>Program</th>
<th>X Server</th>
<th>postgres</th>
<th>pennmush</th>
<th>memcached</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>28k</td>
<td>49k</td>
<td>78k</td>
<td>9k</td>
</tr>
<tr>
<td>Total variables</td>
<td>7795</td>
<td>12350</td>
<td>24372</td>
<td>2350</td>
</tr>
<tr>
<td>Tainted variables</td>
<td>2975 (38%)</td>
<td>5100 (41%)</td>
<td>4168 (17%)</td>
<td>490 (20%)</td>
</tr>
<tr>
<td>Security sensitive variables</td>
<td>823 (10%)</td>
<td>402 (3%)</td>
<td>1573 (6%)</td>
<td>82 (3%)</td>
</tr>
</tbody>
</table>

**Data Structures**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>404</td>
<td>278</td>
<td>311</td>
<td>41</td>
</tr>
<tr>
<td>Sensitive Data structures</td>
<td>61 (15%)</td>
<td>30 (10%)</td>
<td>38 (12%)</td>
<td>7 (17%)</td>
</tr>
</tbody>
</table>

**User-choice Operations**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4760</td>
<td>5063</td>
<td>6485</td>
<td>996</td>
</tr>
<tr>
<td>Sensitive operations</td>
<td>1382 (29%)</td>
<td>1378 (27%)</td>
<td>1382 (21%)</td>
<td>203 (20%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hooks</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>532 (11%)</td>
<td>579 (11%)</td>
<td>714 (11%)</td>
<td>56 (5%)</td>
</tr>
</tbody>
</table>

Programs with manual hook placements
<table>
<thead>
<tr>
<th>Program</th>
<th>X Server</th>
<th>postgres</th>
<th>pennmush</th>
<th>memcached</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>28k</td>
<td>49k</td>
<td>78k</td>
<td>9k</td>
</tr>
<tr>
<td><strong>Total variables</strong></td>
<td>7795</td>
<td>12350</td>
<td>24372</td>
<td>2350</td>
</tr>
<tr>
<td>Tainted variables</td>
<td>2975 (38%)</td>
<td>5100 (41%)</td>
<td>4168 (17%)</td>
<td>1949</td>
</tr>
<tr>
<td>Security sensitive variables</td>
<td>823 (10%)</td>
<td>402 (3%)</td>
<td>1573 (6%)</td>
<td>823</td>
</tr>
<tr>
<td><strong>Data Structures</strong></td>
<td>404</td>
<td>278</td>
<td>311</td>
<td>41</td>
</tr>
<tr>
<td>Sensitive Data structures</td>
<td>61 (15%)</td>
<td>30 (10%)</td>
<td>38 (12%)</td>
<td>7 (17%)</td>
</tr>
<tr>
<td><strong>User-choice Operations</strong></td>
<td>4760</td>
<td>5063</td>
<td>6485</td>
<td>996</td>
</tr>
<tr>
<td>Sensitive operations</td>
<td>1382 (29%)</td>
<td>1378 (27%)</td>
<td>1382 (21%)</td>
<td>203 (20%)</td>
</tr>
<tr>
<td>Hooks</td>
<td>532 (11%)</td>
<td>579 (11%)</td>
<td>714 (11%)</td>
<td>56 (5%)</td>
</tr>
</tbody>
</table>
## Reduction in Effort

<table>
<thead>
<tr>
<th>Program</th>
<th>X Server</th>
<th>postgres</th>
<th>pennmush</th>
<th>memcached</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LOC</strong></td>
<td>28k</td>
<td>49k</td>
<td>78k</td>
<td>9k</td>
</tr>
<tr>
<td><strong>Total variables</strong></td>
<td>7795</td>
<td>12350</td>
<td>24372</td>
<td>2350</td>
</tr>
<tr>
<td>Tainted variables</td>
<td>2975 (18%)</td>
<td>5100 (41%)</td>
<td>4168 (17%)</td>
<td>490 (20%)</td>
</tr>
<tr>
<td>Security sensitive variables</td>
<td>823 (10%)</td>
<td>402 (3%)</td>
<td>1573 (6%)</td>
<td>82 (3%)</td>
</tr>
<tr>
<td><strong>Data Structures</strong></td>
<td>404</td>
<td>278</td>
<td>311</td>
<td>41</td>
</tr>
<tr>
<td>Sensitive Data structures</td>
<td>61 (15%)</td>
<td>30 (10%)</td>
<td>38 (12%)</td>
<td>7 (17%)</td>
</tr>
<tr>
<td><strong>User-choice Operations</strong></td>
<td>4760</td>
<td>5063</td>
<td>6485</td>
<td>996</td>
</tr>
<tr>
<td>Sensitive operations</td>
<td>1382 (29%)</td>
<td>1378 (27%)</td>
<td>1382 (21%)</td>
<td>203 (20%)</td>
</tr>
<tr>
<td>Hooks</td>
<td>532 (11%)</td>
<td>579 (11%)</td>
<td>714 (11%)</td>
<td>56 (5%)</td>
</tr>
</tbody>
</table>
Placement Comparison

- X Server:
  - Manual: 201 hooks
  - Automated: 532 hooks

- Postgres:
  - Manual: ~370
  - Automated: 579

What does this mean?
Hoist: Op Dominance

```
read(pgcSrc->planemask)
read(pgcSrc->fgPixel)
read(pgcSrc->alu)
read(pgcSrc->bgPixel)
```
Remove: Object Dominance

\[
\text{Resource } \text{res} = \text{ClientTable}[i]
\]

\[
\text{WindowPtr } * \text{pWin} = (\text{WindowPtr } *) \text{res}
\]

\[
\text{WindowPtr } * \text{pChild} = \text{pWin} \rightarrow \text{firstChild} \rightarrow \text{nextSib}
\]

\[
\text{pChild} \rightarrow \text{mapped} = \text{True}
\]
Automated Hook Per Manual

![Bar Chart: Frequency of No. of Dominated Automated Hooks]

- X-axis: No. of Dominated Automated Hooks
- Y-axis: Frequency

The chart shows the distribution of automated hooks per manual, with a peak at 45 automated hooks.
DIFC Programs by
Automatic Instrumentation

William Harris, Somesh Jha, and Thomas Reps
DIFC Mechanics

{ a } -> OS -> P3

{ a }
Challenge of Instrumentation

Instrument DIFC code that is:

1. Legal
2. Secure
3. Functional
Key Insight

From a program and **policy**, an instrumenter statically **constrains** labels to instrument **DIFC code**.
Key Payoffs of Constraints

- Naturally express *semantics, policies*

- Efficiently generate DIFC code

- Provide useful debugging information
Key Payoffs of Constraints
void Spawner() {

    Conn c = requestConn();

1:  spawn(Worker, c);

2:  }

Spawner

Requester

Lab 1  Pos 1  Neg 1  Create 1

Lab 2  Pos 2  Neg 2  Create 2

Worker

Network
Legal Rule #1:

A process’s label only increases by tags in its positive capability.

1: Conn c = requestConn();
2: spawn(Worker, c);

Lab2 subseteq (Lab1 U Pos1)
Legal Rule #2:

A process’s label only decreases by tags in its negative capability.

1: Conn c = requestConn();
2: spawn(Worker, c);

Lab2 superseteq (Lab1 - Neg1)
Legal Rule #3:

A process’s capabilities only increase to hold tags that the process creates.

1: Conn c = requestConn();
2: spawn(Worker, c);

\[
\text{Pos2 subseteqq (Pos1 U Create1)} \\
\text{Neg2 subseteqq (Neg1 U Create1)}
\]
Solving Constraints

• NP-complete in general

• Amenable to SMT solvers in practice
void Spawner() {

    Lab1   Pos1   Neg1   Create1
    { }     { }    { }    { a }

tag a = create_tag();

1:    Conn c = requestConn();

    Lab2   Pos2   Neg2   Create2
    { a }   { a }    { }    { }

2:    spawn(Worker, c, lab: { a }, pos: { a }, neg: { });

}
Program Retrofit

• Start with program
  ‣ And a minimal number of other inputs…
• Security goal
• Trust and threat models
• Reference monitor
• Policy
  ‣ Hook placement
  ‣ DIFC weaving
Take Away

• Programs are not written for security up-front
• Retrofitting with security manually is error-prone
  ‣ May even be expensive
• We examined two techniques for retrofitting programs with security code automatically
  ‣ Can we change the task of writing security code?
  ‣ Not for programmers…