CMPSC 497: Reference Monitors

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

- Manual programming of security code often leads to errors
  - Even expert programmers made mistakes in adding authorization for the Linux Security Modules project

- What are requirements for a correct authorization mechanism?
  - How can we enable programmers to satisfy those requirements?
Authorization

- How is authorization integrated into an existing (host) program?
Authorizing Access

Resource user

Operation request

Resource manager

Allowed?

Reference monitor

Response

<Alice, /etc/passwd, File_Read>
Authorizing Access

Resource user

Operation request

Resource manager

Authorization Hooks

Reference monitor

Allowed?

Authorization policy

Response

YES/NO
Authorization

• How is authorization integrated into an existing (host) program?

• Authorization system
  ▸ Authorization hooks – constructs authorization queries (subject, object, operation) and invokes reference monitor
    • Integrated into host program
  ▸ Reference monitor module – processes authorization queries into Y/N decisions using authorization policy
  ▸ Authorization policy – essentially a database relating subjects and objects to the operations that subjects are authorized to perform on objects
Requirements

• What should be the requirements for an authorization system to correctly enforce an authorization policy?
Reference Monitor Concept

- Reference monitor concept was defined in 1972 by James Anderson to describe design requirements on a “reference validation mechanism” (read authorization system)
  - The reference validation mechanism must always be invoked (complete mediation).
  - The reference validation mechanism must be tamperproof (tamperproof).
  - The reference validation mechanism must be small enough to be subject to analysis and tests, the completeness of which can be assured (verifiable).
Reference Monitor Concept

• So what do these mean?
  ‣ The reference validation mechanism must always be invoked (complete mediation).
  ‣ The reference validation mechanism must be tamperproof (tamperproof).
  ‣ The reference validation mechanism must be small enough to be subject to analysis and tests, the completeness of which can be assured (verifiable).
Complete Mediation

• So what do these mean?
  ‣ The reference validation mechanism must always be invoked (complete mediation)

• A program has security-sensitive operations, and the authorization system must be invoked to control subjects’ access to objects in such operations
Tamperproof

• So what do these mean?
  ‣ The reference validation mechanism must be tamperproof (tamperproof).

• The authorization system code, including modules and hooks, and data, including authorization policies, must only be writeable by trusted subjects
  ‣ Ideally, such code and data should be set when the program is initiated (e.g., OS is booted) and remain unchanged throughout its execution
Verifiable

- So what do these mean?
  - The reference validation mechanism must be small enough to be subject to analysis and tests, the completeness of which can be assured (verifiable).

- The authorization system’s code should be tested comprehensively to validate its correctness

- The authorization system’s policy should be evaluated to validate that it enforces security correctly
Complete Mediation

• What does this *really* mean?
  ‣ The reference validation mechanism *must always be invoked* (complete mediation)
  ‣ A program has *security-sensitive operations*, and the authorization system must be invoked to control subjects’ access to objects in such operations

• How do we ensure that programmers satisfy this requirement?
  ‣ In particular, what is a “security-sensitive operation?”
Security-Sensitive Operations

Program

Challenges

- Identifying Operations:
  - Set of statements
Security-Sensitive Operations

Challenges

• Identifying Operations:
  ‣ Set of statements
  ‣ Only a subset are security-sensitive
  ‣ May be a set of statements
• What makes a set of statements a security-sensitive operation?
Security-Sensitive Objects

Challenges

- Identifying Security-Sensitive Objects
Challenges

• Identifying Security-Sensitive Objects:
  ‣ Programs manipulate many variables
    • 7800 in X server
    • Of over 400 data types
Identify SSOs in Programs

- By statements
  - List the code snippets ("fingerprints") that imply security-sensitive operations

- By types
  - List the data types (structures) of variables that imply security-sensitive objects
  - Structure member access – “struct A” is security-sensitive; variable “x” is of type “struct A”, so “x→field” is a security-sensitive operation
Structure Member Access

• For example, control access to “struct file”
  ‣ All structure member access should be proceeded mediation

• Found missing hooks

Security check

```c
linux/fs/read_write.c:
ssize_t vfs_read(...) {
  ...
  ret = security_file_permission(file, ...);
  if (!ret) {
    ret = file->f_op->read(file, ...);
  }
  ...
}
```

```c
linux/fs/readdir.c:
ssize_t vfs_readdir(...) {
  ...
  ret = security_file_permission(file, ...);
  if (!ret) {
    ret = file->f_op->readdir(file, ...);
  }
  ...
}
```

```c
linux/fs/read_write.c:
ssize_t do_readv_writer(...) {
  ...
  ret = file->f_op->readv(file, ...);
  ...
}
```

Same security sensitive operation: file_read/write
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 → Retrofitted code

INSECURE → SECURE
Hook Placement Problem

Program

Challenges

• What is ideal placement?
  ▶ Minimal number of hooks?
  • May block legitimate function
  • Need to be authorized to perform both operations if only first hook is used
Hook Placement Problem

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.
Idea: Request Choices

• In servers, \textit{user-request} determines \textit{choices} that client subjects can make in the program

• “Choice”:
  ‣ Determines which \textit{objects} are selected to be operated upon (data-flow choices)
  ‣ Determines which \textit{operation} is selected to be performed on objects (control-flow choices)
Inputs and Containers

Program

A

B

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

D

E

F

I

K

H

J

L

Container \( O \)

- \( o1 \)
- \( o2 \)
- \( o3 \)
- \( o4 \)

Request Interface \( i \)

User A

User B
Using Taints for Lookup

Using Taints for Lookup

Using Taints for Lookup
Security-Sensitive Object

user A

Request Interface $i$

user B

Program

A

$C \leftarrow \text{Lookup}(O,i)$

User B

Container $O$

$O_1$

$O_2$

$O_3$

$O_4$

Op 1.0

D

E

F

I

H

J

K

L

Security-Sensitive Object
Tainted Predicate

Control statement predicated on tainted variable
User-Choice Operations

Program

A

Request Interface

B

C

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

User A

User B

Op 1.0

Op 1.1

Op 1.3

D

E

F

H

I

J

K

L

read v

User choice operation

write v
Security-Sensitive Operations

User A
User B

Request Interface \( i \)

Program

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

Container \( O \)

Security-sensitive operation

Op 1.0

Op 1.1

Op 1.3

Op 1.3

\( \text{read } v \)

\( \text{write } v \)
Don’t Need to Mediate All

• No need to mediate redundant operations
  ‣ If one operation performs the same structure member accesses as a prior, mediated operation then no need to mediate again
    • First operation (A→f) and second operation (A→f)

• No need to mediate if already enforce expected policy
  ‣ Already blocked all the unauthorized subjects in all cases, then no need for further mediation
    • Only subject X is authorized for first operation and second op
Hoisting

```
Op 1.1
pProp->name = property
pProp->format = format
pProp->data = data
pProp->size = len
```

```
Op 2
(mode==REPLACE)
```

```
Op 2.1
pProp->format = format
pProp->data = data
pProp->size = len
```

```
Op 2.2
pProp->data = data
pProp->size = len
```

```
Op 2
(mode==APPEND)
```

```
Op
Structure member access to mediate
```

```
Connection from outside this procedure
```

```
control statement
```

```
Dummy node
```

```
Entry
dxChangeWindowProperty
```

```
write(pProp->size)
write(pProp->data)
```

```
(re==BadMatch)
```
Removal

Op  Structure member
Op 1.1  access to mediate
Op 1.2  Connection from outside
this procedure
Op 2  control
Op 2.1  statement
Op 2.2  Dummy node
Op 2.3  Operation

write(pProp->format)
write(pProp->name)
write(pProp->format)
write(pProp->size)
write(pProp->data)

Entry
(dixChangeWindowProperty)

write(pProp->size)
write(pProp->data)

(rc==BadMatch)

Op 2.1
pProp->format = format
pProp->data = data
pProp->size = len

Op 2.2
pProp->format = format
pProp->data = data
pProp->size = len

Op 2
(mode==REPLACE)

Op 1.1
pProp->name = property
pProp->format = format
pProp->data = data
pProp->size = len

Op 1.2
pProp->format = format
pProp->data = data
pProp->size = len

Connection from outside
this procedure

Systems and Internet Infrastructure Security Laboratory (SIIS)
Placement

Op
Structure member access to mediate

Connection from outside this procedure

control statement

Dummy node

Op 1.1
pProp->name = property
pProp->format = format
pProp->data = data
pProp->size = len

Op 2
Op 2.1
pProp->format = format
pProp->data = data
pProp->size = len

Op 2.2
pProp->data = data
pProp->size = len

Op 1.1
write(pProp->name)
pProp->name = property

Op 2
write(pProp->format)
pProp->format = format

Op 2.1
write(pProp->data)
pProp->data = data

Op 2.2
write(pProp->size)
pProp->size = len

Entry (-disChangeWindowProperty)
Placement Comparison

• X Server:
  ‣ Manual: 201 hooks
  ‣ Automated: 532 hooks

• Postgres:
  ‣ Manual: ~370
  ‣ Automated: 579

What does this mean?
For Operations

read(pgcSrc->planemask)  read(pgcSrc->fgPixel)  read(pgcSrc->alu)  ...  read(pgcSrc->bgPixel)

Hoist
For Objects

Resource res = ClientTable[i]

WindowPtr * pWin = (WindowPtr *) res

WindowPtr * pChild = pWin->firstChild->nextSib

pChild->mapped = True

Remove
Allowed Subjects per Op

- \( \text{Allowed}(o) \): Subset of subjects in \( U \) that are allowed to perform operation \( o \).

- **Constraint 1:**
  - \( \text{Allowed}(o_1) = \text{Allowed}(o_2) \), then \( o_1 \) equals \( o_2 \)

- **Constraint 2:**
  - \( \text{Allowed}(o_1) \subset \text{Allowed}(o_2) \), then \( o_1 \) subsumes \( o_2 \)
Already Blocked

• Suppose we have two operations, A and B
  ‣ First, code performs A followed by B in all cases
  ‣ Suppose operation A accesses object J and operation B accesses object K
  ‣ Suppose there are two subjects, X and Y

• If we have the following access control policy
  ‣ (X, J) (X, K), (Y, K) meaning (S, O) where subject S can access object O

• Do we need an authorization hook at B?
Already Blocked

• Suppose we have two operations, A and B
  ‣ First, code performs A followed by B in all cases
  ‣ Suppose operation A accesses object J and operation B accesses object K
  ‣ Suppose there are two subjects, X and Y

• If we have the following access control policy
  ‣ (X, J) (X, K), (Y, K) meaning (S, O) where subject S can access object O

• Do we need an authorization hook at B? No
Already Blocked

- Do we need an authorization hook at B?
  - No. Why not?
Already Blocked

- Do we need an authorization hook at B?
  - No. Why not?

- All subjects “allowed” at A (subject X) are allowed to access any object at B (only object K)
  - So, further mediation is not necessary to enforce this policy

- In general, programs place authorization hooks to enforce the policies they have in mind
  - Although currently this is implicit
Take Away

• For a program to enforce access control correctly, it needs to satisfy the reference monitor concept
  ‣ Complete Mediation, Tamperproof, and Verifiable

• However, the reference monitor concept is not precisely defined
  ‣ Complete mediation depends on the identification of security-sensitive operations

• Automated techniques to place hooks based on choice – still require some knowledge of policies