Runtime Analysis

November 28, 2011
Analysis So Far

• Prove whether a property always holds
  ‣ *May analysis*

• Prove whether a property can hold
  ‣ *Must analysis*

• Key step: abstract interpretation to overapproximate behavior of program

• But, it can be expensive and complex
Runtime Analysis

- Collect traces of program runs to evaluate a property
- Testing
  - Run test cases to determine if property holds (or fails to hold) in all cases
  - Inherently incomplete
- Traces
  - Compare several runs to determine if a property holds across runs
  - Incomplete?
Example

- Runtime Verification of Authorization Hook Placement for the Linux Security Modules Framework
- Linux Security Modules (LSM) framework
- Problem: Are authorization hooks placed correctly?
  - What does that mean?
Mediation

- **Security-sensitive Operations**: These are the operations that impact the security of the system.

- **Controlled Operations**: A subset of security-sensitive operations that mediate access to all other security-sensitive operations. These operations define a mediation interface.

- **Authorization Hooks**: These are the authorization checks in the system (e.g., the LSM-patched Linux kernel).

- **Policy Operations**: These are the conceptual operations authorized by the authorization hooks.
Mediation Overview

System Call Approach

User

Syscall Trap

H

P/C

Kernel

H: Authorization Hook
P: Policy Operation
C: Controlled
Operation
S: Security-sensitive
Operation

LSM Approach

H

P

C

C

S

S

S

S

S

...
Security-Sensitive Ops

• What code-level operations indicate security-sensitivity?

• Variable access?

• Structure member access?

• Global access?
Key Challenges

• **Identify Controlled Operations**: Find the set of security-sensitive operations that define a mediation interface

• **Determine Authorization Requirements**: For each controlled operation, identify the policy operation

• **Verify Complete Authorization**: For each controlled operation, verify that the correct authorization requirements (policy operation) is enforced

• **Verify Hook Placement Clarity**: Controlled operations implementing a policy operation should be easily identifiable from their authorization hooks
Key Relations

- **Authorization Hook** predicts **Policy Operation**
- **Policy Operation** defines **Controlled Operation**
- **Controlled Operation** mediates **Security-sensitive Operation**

1. mediates
2. defines
3. mediates
4. predicts
Analysis Approach

- Check consistency between hooks and security-sensitive operations
  - Traces
- Sensitivity
  - Structure member accesses
  - Hooks
- Consistent relationship indicates hook is associated with SMAs (make a controlled op)
  - Sensitivity can vary in granularity
## Sensitivities

<table>
<thead>
<tr>
<th>Factor</th>
<th>Authorizations are same for:</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Call</td>
<td>all controlled operations in system call</td>
</tr>
<tr>
<td>Syscall Inputs</td>
<td>all controlled operations in same system call with same inputs</td>
</tr>
<tr>
<td>Datatype</td>
<td>all controlled operations on objects of the same datatype</td>
</tr>
<tr>
<td>Object</td>
<td>all controlled operations on the same object</td>
</tr>
<tr>
<td>Member</td>
<td>all controlled operations on same datatype, accessing same member, with same operation</td>
</tr>
<tr>
<td>Function</td>
<td>all same member controlled operations in same function</td>
</tr>
<tr>
<td>Intra-function</td>
<td>same controlled operation instance</td>
</tr>
<tr>
<td>Path</td>
<td>same execution path to same controlled operation instance</td>
</tr>
</tbody>
</table>

**Table 1: Authorization Sensitivity Factors: names and effects on authorizations**
Anomalies

• For SMAs to be a controlled op
  ‣ Path: all traces with SMA should have same hooks
    • Not dependent on paths taken to get there
  ‣ Function: all traces with same SMA type in same function should have same hooks
    • SMA in function defines controlled op if always associated with hook
Implementation

• Propose sensitivity rules for system call processing
  ‣ Propose relationship between hooks and controlled ops

• Log traces of system call processing
  ‣ Collect syscall entry/exit/args, function entry/exit, controlled ops, and hooks

• Compute whether hooks always/sometimes/never in trace for each controlled op
  ‣ Evaluate whether the current sensitivity rules express the expected consistency

• Update sensitivity rules
Implementation
Logging

• Authorization hooks
  ‣ LSM itself

• Controlled operations (SSOs)
  ‣ GCC module

• Control data
  ‣ GCC flag

• System call contexts
  ‣ Kernel scheduling loop
Log Filtering Rules

- For sensitivity
  - Filter log entries processed to determine sensitivity

```
# Path sensitive rule for operation at 0xc014f046
1 = (+,id.type,CONTEXT) (+,di.cfm.eax,READ)
2 (D,1) = (+,id.type,CNTL.OP)
(+,di.dfm.ip,0xc014f046)
3 (D,1) = (+,id.type,SEC.CHI)

# Member sensitive rule for inode member i.flock read access
1 = (+,id.type,CONTEXT) (+,di.cfm.eax,READ)
2 (D,1) = (+,id.type,CNTL.OP)
(+,di.dfm.class,OT.INODE)
(+,di.dfm.member,i.flock)
(+,di.dfm.access,OP.READ)
3 (D,1) = (+,id.type,SEC.CHI)

# Input sensitive rule for open for read access, but not path.walk
1 = (+,id.type,CONTEXT) (+,di.cfm.eax,OPEN)
(+,co ecx,RONLY)
2 (D,1) = (+,id.type,FUNC)
(+,di.fmm.ip,path.walk)
3 (D,1)(N,2) = (+,ALL,0,0)
```

Figure 4: Example authorization sensitivity filtering rules
Results

• Missing hook
  ‣ Setgroups16

• Have different numbers of hooks
  ‣ Fcntl (set_fowner)

• Missing hook
  ‣ Fcntl (signal)

• Missing hook
  ‣ Read (Memory mapped files)
Runtime Analysis

• Choose test cases
• Collect traces (content of traces)
• Analyze traces
• Evaluate property
Hook Placement

• A variety of analysis for hook placement and testing
  • Zhang [USENIX 2002]
  • Ganapathy [CCS 3005, Oakland 2006, ICSE 2007]
  • Tan [USENIX 2008]
  • [AsiaCCS 2008]
  • Son [OOPSLA 2010]
  • King etal [ESOP 2010]
• We are working on a purely static analysis