Detecting Manipulated Remote Call Streams

*Jonathon Giffin, Somesh Jha, Barton Miller*

Computer Sciences Department
University of Wisconsin

giffin@cs.wisc.edu
Intrusion Detection and Specification-Based Monitoring

• The Condor attack
  How to easily do dangerous and malicious things to a running job

• Binary analysis
  How to detect attempted intrusions with pre-execution static analysis and runtime monitoring

• Program instrumentation
  How to improve model precision & performance
## Intrusion Detection

**Goal:** Discover attempts to gain malicious access to a system

### Specification-Based Monitoring
- Specify constraints upon program behavior
- Ensure execution does not violate specification
- Our work; Ko, et al.
- Specifications can be cumbersome to create

### Misuse Detection
- Specify patterns of attack or misuse
- Ensure misuse patterns do not arise at runtime
- Snort
- Rigid: cannot adapt to novel attacks

### Anomaly Detection
- Learn typical behavior of application
- Variations indicate potential intrusions
- IDES
- High false alarm rate

---

<table>
<thead>
<tr>
<th>Misuse Detection</th>
<th>Specification-Based Monitoring</th>
<th>Anomaly Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Specify patterns of attack or misuse</td>
<td>• Specify constraints upon program behavior</td>
<td>• Learn typical behavior of application</td>
</tr>
<tr>
<td>• Ensure misuse patterns do not arise at runtime</td>
<td>• Ensure execution does not violate specification</td>
<td>• Variations indicate potential intrusions</td>
</tr>
<tr>
<td>• Snort</td>
<td>• Our work; Ko, et al.</td>
<td>• IDES</td>
</tr>
<tr>
<td>• Rigid: cannot adapt to novel attacks</td>
<td>• Specifications can be cumbersome to create</td>
<td>• High false alarm rate</td>
</tr>
</tbody>
</table>
Specification

Analyst or Administrator

Training Sets

Static Source Code Analysis

Static Binary Code Analysis

Execution Obeys Static Rule Set

Execution Matches Model of Application

Enforcement
Our Approach: Specification

Static analysis of binary code

• Specifications are automatically generated

• Not reliant upon analysts to produce accurate specifications

• Analyzes all execution paths

• Source code may be unavailable
Our Approach: Enforcement

Operate an automaton modeling correct system call sequences

• Dynamic ruleset
Technical Contributions

• Binary analysis
• Model comparisons
• Techniques to improve precision
  - Null call insertion
  - Call site renaming
• Techniques to improve performance
  - Stack abstractions
  - Null call insertion: Practical results using push-down automaton (PDA) models
Example: The Condor Attack

- Users dispatch programs for remote execution
- Remote jobs send critical system calls back to local machine for execution
Example: The Condor Attack

- Attackers can manipulate remotely executing program to gain access to user’s machine

![Diagram showing the Condor Attack]
Countering Remote Attacks

- **Goal:** Even if an intruder can see, examine, and fully control the remote job, no harm can come to the local machine.

- **Method:** Model all possible sequences of remote system calls. At runtime, update the model with each received call.

- **Key technology:** Static analysis of binary code.
Execution Monitoring

Checking Shadow

system calls

Modified User Job

Submitting Host

Execution Host

Job Model
Execution Monitoring

Submitting Host

Checking Shadow

Modified User Job

Execution Host

system calls

FAIL

Job Model

Call 2
Model Construction

Analyzer

User Job

Checking Shadow

Modified User Job

Binary Program

Control Flow Graphs

Local Automata

Global Automaton
The Binary View (SPARC)

function:
  save %sp, 0x96, %sp
  cmp %i0, 0
  bge L1
  mov 15, %o1
  call read
  mov 0, %o0
  call line
  nop
  b L2
  nop
L1:
  call read
  mov %i0, %o0
  call close
  mov %i0, %o0
L2:
  ret
  restore

function (int a) {
  if (a < 0) {
    read(0, 15);
    line();
  } else {
    read(a, 15);
    close(a);
  }
}
Control Flow Graph Generation

function:
  save %sp, 0x96, %sp
  cmp %i0, 0
  bge L1
  mov 15, %o1
  call read
  mov 0, %o0
  call line
  nop
  b L2
  nop
L1:
  call read
  mov %i0, %o0
  call close
  mov %i0, %o0
L2:
  ret
  restore
Control Flow Graph Translation

read → read → close → line → read

CFG ENTRY
  bge
  call read
  call read
  call close
  call line

ret
CFG EXIT
Interprocedural Model Generation

A

read

read

line

ε

B

close

line

e

line

e

write

ε

close

Possible Paths

A

read

read

ε

ε

write

B

close

close
Possible Paths

[A] read → read

ε → close

line

write

ε → close

B
Impossible Paths

A

read

read

line

write

ε

close

close

B

ε

ε
Impossible Paths

A

B

read

read

line

write

close

close
Adding Context Sensitivity

read
read
line
write
push
X
push
X
pop
X
pop
Y
pop
Y

PDA State Explosion

• $\varepsilon$-edge identifiers maintained on a stack
  - Stack may grow to be unbounded

• Solution:
  - Bound the maximum size of the runtime stack
  - A regular language overapproximation of the context-free language of the PDA
Prototype Implementation

• Simulates remote execution environment
• Measure model precision
• Measure runtime overheads
• Measure the effect of changing maximum stack depth on bounded PDA model
## Test Programs

<table>
<thead>
<tr>
<th>Program Size in Instructions</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip</td>
<td>Compress a 13 MB file</td>
</tr>
<tr>
<td>GNU finger</td>
<td>Finger 3 non-local users</td>
</tr>
<tr>
<td>procmail</td>
<td>Process 1 incoming email message</td>
</tr>
<tr>
<td>56,686</td>
<td></td>
</tr>
<tr>
<td>95,534</td>
<td></td>
</tr>
<tr>
<td>107,167</td>
<td></td>
</tr>
</tbody>
</table>
Precision Metric

• Average branching factor

• Lower values indicate greater precision
Precision: NFA Model

Average Branching Factor

Program

gzip
GNU finger
procmail

None Rename Argument Capture Rename+Capture
Optimizations to Improve Precision

• Observation: PDA is more precise than NFA because it provides context sensitivity

• Idea: Insert null calls into NFA model to add some context sensitivity without suffering runtime cost of PDA
Null Call Experiments

• Inserted null calls at 3 rates
  - High: At entries of functions with fan-in of 2 or greater
  - Medium: At entries of functions with fan-in of 5 or greater
  - Low: At entries of functions with fan-in of 10 or greater
Precision: NFA Model with Null Calls

Program

<table>
<thead>
<tr>
<th>Program</th>
<th>Rename+Capture</th>
<th>Low Insertion Rate</th>
<th>Medium Insertion Rate</th>
<th>High Insertion Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GNU finger</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>procmail</td>
<td>13</td>
<td>10</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>
Precision: PDA Model with Null Calls, procmail

- Average Branching Factor
- Stack Bound

Graph showing the relationship between stack bound and average branching factor for different insertion rates and a Rename+Capture method.
Overhead: PDA Model with Null Calls, procmail

Stack Bound

Overhead (seconds)

- High Insertion Rate
- Medium Insertion Rate
- Low Insertion Rate
- Rename+Capture

750 s 95 s

0 1 2 3 4 5 6 7 8 9 10

Important Ideas

• Specifications generated automatically from binary code analysis
• Operate a finite state machine modeling correct execution
• PDA model is precise but suffers high overhead
• Bounded PDA stack & null calls allow use of precise PDA model
Detecting Manipulated Remote Call Streams

Jonathon Giffin, Somesh Jha, Barton Miller
Computer Sciences Department
University of Wisconsin
giffin@cs.wisc.edu