Protecting Your Local Host From Remote Security Attacks

Jonathon Giffin, Somesh Jha, Bart Miller
University of Wisconsin
Overview

• How to Easily do Dangerous and Malicious Things to a Running Job

• How to Detect Manipulations of Your Jobs with Pre-Execution Static Analysis and Runtime Monitoring
A New View

• Running programs are objects to be easily manipulated

• The vehicle: the **DynInst API**
DynInst: Dynamic Instrumentation

• Machine independent library for instrumentation of running processes
• Modify control flow of the process:
  - Load new code into the process
  - Remove, replace, or redirect function calls
  - Asynchronously call any function in the process
Condor Attack: Lurking Jobs

- **Shadow Process**: `giffin`
- **Malicious User Job**: `nobody`

**Submitting Host** → **system calls** → **Execution Host**
Condor Attack: Lurking Jobs

- Shadow Process: *giffin*
- System calls
- Execution Host
- Malicious User Job: *nobody*
- Lurker Process: *nobody*
- Submitting Host

21 March 2002
WiSA - Jonathon Giffin
Condor Attack: Lurking Jobs

Submitting Host  Execution Host

Lurker Process

nobody
Condor Attack: Lurking Jobs

- Shadow Process: bart
  - Submitting Host

- Innocent User Job: nobody
  - Execution Host

System calls from Shadow Process to Innocent User Job.
Condor Attack: Lurking Jobs

Shadow Process

\textit{bart}

\hspace{2cm}

system calls

\hspace{2cm}

Innocent User Job

\textit{nobody}

\hspace{2cm}

Lurker Process

\textit{nobody}

Submitting Host

\hspace{1cm}

Execution Host

\hspace{1cm}

attach
Condor Attack: Lurking Jobs

Shadow Process

Innocent User Job

Lurker Process

Submitting Host

Execution Host

system calls

Control remote system calls

attach
Condor Attack: Lurking Jobs

Submitting Host

Shadow Process

`bart`

`rm -rf *`

Execution Host

Innocent User Job

`nobody`

Attachment

Lurker Process

`nobody`

Control remote system calls

System calls

21 March 2002 WiSA - Jonathon Giffin
Can We Safely Execute Our Jobs Remotely?

The threats:
1. Cause the job to make improper remote system calls.
2. Cause the job to calculate an incorrect answer.
3. Steal data from the remote job.

Threat protection strategies:
- Monitor execution of remote job (threat #1)
- File or system call sand-boxing (#1)
- Obfuscate or encode remote job (#1, #3)
- Replicate remote job (#2)
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

User Job

Analyzer

Checking Shadow

Modified User Job
Execution Monitoring

Submitting Host

Checking Shadow

Job Model

system calls

Execution Host

Modified User Job
Execution Monitoring

Submitting Host

Modified User Job

Execution Host

Checking Shadow

system calls

Job Model

Call 3
Model Construction

Binary Program → Control Flow Graphs → Local Automata → Global Automaton

User Job
Analyzer
Checking Shadow
Modified User Job
function( int a ) {
    if( a < 0 ) {
        read( 0, 15 );
        line();
    } else {
        read( a, 15 );
        close( a );
    }
}
Control Flow Graph Translation

CFG ENTRY

if

call read

call close

return

CFG EXIT

read read

close line
Interprocedural Model Generation
Interprocedural Model Generation

21 March 2002

WiSA - Jonathon Giffin
Interprocedural Model Generation

A

read

read

close

ε

B

line

write

close

ε

ε

21 March 2002

WiSA - Jonathon Giffin
Interprocedural Model Generation

A

read

read

ε

ε

ε

ε

ε

ε

B

line

write

close

close
Possible Paths

A
read
read
close

B
line
write

ε ε ε ε ε close
Possible Paths

A

read
read

ε

ε

ε

ε

ε

B

write
write

line

close

close
Impossible Paths

A
read
read

B
write
close
close

ε

21 March 2002
WiSA - Jonathon Giffin
Impossible Paths

A

read

read

ε

ε

ε

ε

line

write

close

B

ε

close

21 March 2002

WiSA - Jonathon Giffin
Adding Context Sensitivity

A

read

read

B

line

write

close

close
Rewriting User Job

User Job

Analyzer

Checking Shadow

Modified User Job

Binary Program

Rewritten Binary
Null Call
Insertion

A

read

read

close

B

line

write

close

ε

ε

ε

ε

ε

ε
Null Call
Insertion

A

read

read

null_1

null_2

close

B

line

write

ε

ε

null_2

close

21 March 2002

WiSA - Jonathon Giffin
Call Site Renaming

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

- Give each monitored call site a unique name
- Captures arguments
- Obfuscation
- Limits attack call set
- Reduces nondeterminism
Call Site Renaming

function( int a ) {
    if( a < 0 ) {
        _638();
        line();
    } else {
        read( a, 15 );
        close( a );
    }
}

• Give each monitored call site a unique name
• Captures arguments
• Obfuscation
• Limits attack call set
• Reduces nondeterminism
Call Site Renaming

function( int a ){
    if( a < 0 ) {
        _638();
        line();
    } else {
        _83( a );
        close( a );
    }
}

- Give each monitored call site a unique name
- Captures arguments
- Obfuscation
- Limits attack call set
- Reduces nondeterminism
Call Site Renaming

function( int a ) {
    if( a < 0 ) {
        _638();
        line();
    } else {
        _83( a );
        _1920( a );
    }
}

• Give each monitored call site a unique name
• Captures arguments
• Obfuscation
• Limits attack call set
• Reduces nondeterminism
Call Site Renaming

- Give each monitored call site a unique name
- Captures arguments
- Obfuscation
- Limits attack call set
- Reduces nondeterminism
Prototype Implementation

- Simulates remote execution environment
- Null calls inserted at function entries
- Measure null call overheads
- Measure null call bandwidth use
- Measure model precision
## Test Programs

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

- Average branching factor
Optimizations Improve Precision

The graph shows the average branching factor for different programs and optimization techniques. The programs include gzip, GNU finger, and procmail. The optimization techniques are None, Rename, Argument Capture, and Rename+Capture. The graph indicates that optimization techniques can significantly improve precision, with Rename+Capture showing the highest average branching factor for procmail.
## Null Call Costs: Monitoring Overhead & Bandwidth

<table>
<thead>
<tr>
<th>Insertion Rate</th>
<th>High</th>
<th>Medium</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip</td>
<td>747.0 %</td>
<td>&lt; 0.1 %</td>
<td>&lt; 0.1 %</td>
</tr>
<tr>
<td>GNU finger</td>
<td>0.1 %</td>
<td>0.1 %</td>
<td>&lt; 0.1 %</td>
</tr>
<tr>
<td>procmail</td>
<td>0.8 %</td>
<td>1.1 %</td>
<td>0.7 %</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Insertion Rate</th>
<th>gzip</th>
<th>GNU finger</th>
<th>procmail</th>
</tr>
</thead>
<tbody>
<tr>
<td>4350.0 Kbps</td>
<td>5.6 Kbps</td>
<td>0.0 Kbps</td>
<td></td>
</tr>
<tr>
<td>14.1 Kbps</td>
<td>9.1 Kbps</td>
<td>0.9 Kbps</td>
<td></td>
</tr>
<tr>
<td>18.2 Kbps</td>
<td>13.1 Kbps</td>
<td>4.0 Kbps</td>
<td></td>
</tr>
</tbody>
</table>
Precision Improves with Null Calls

Program

- gzip
- GNU finger
- procmail

Average Branching Factor

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

21 March 2002  WiSA - Jonathon Giffin

43
Important Ideas

- Running jobs remotely enables malicious attacks against the shadow process.
- Pre-execution static analysis to construct a model of the remote call sequences addresses this threat.
- Improving model precision is a hard problem. Null calls are an effective and reasonable optimization.
Protecting Your Local Host From Remote Security Attacks

Jonathon Giffin, Somesh Jha, Bart Miller
University of Wisconsin