Pervasive Detection of Thread-Process Races In Deployed Systems

Columbia University
Oren Laadan
Nicolas Viennot
Chia-Che Tsai
Chris Blinn
Junfeng Yang
Jason Nieh
ps aux | grep pizza
ps aux | grep pizza outputs how many lines:

A) 0
B) 1
C) it depends
D) I can't think, you made me hungry with the pizza thing
ps aux | grep pizza
outputs how many lines:

A) 0
B) 1
C) it depends
D) I can't think, you made me hungry with the pizza thing
ps aux | grep pizza
ps aux | grep pizza

shell

$ ps aux | grep pizza
`ps aux | grep pizza`

$ ps aux | grep pizza
ps aux | grep pizza
$ ps aux | grep pizza
$ ps aux | grep pizza
nviennot  3 ... S+ 13:30  0:00 grep pizza
$
```bash
$ ps aux | grep pizza
$
That's a process race
Process Races

- **Process races** occur when multiple processes access shared resources (such as files) without proper synchronization.

- Examples:
  - parallel make (make -j) failure
  - ps aux | grep pizza
ps aux | grep xxx

Bug 54127 - /usr/bin/licq has a race condition

Status: CLOSED DUPLICATE of bug 55057

Aliases: None (edit)

Product: Red Hat Linux
Component(s): licq (Show other bugs)
Version(s): 7.3
Platform: i386 Linux

Priority: medium Severity: medium
Process Races Are Numerous

- Searched for “race” in the distro bug trackers (Ubuntu, Redhat/Fedora, Gentoo, Debian, CentOS)
- 9000+ results
- Sampled 500+ of them
- 109 unique bugs due to process races
Process Races Are Dangerous

Source: samples from Ubuntu, Redhat, Fedora, Gentoo, Debian, CentOS bug trackers
### Process Races Are Hard To Detect

<table>
<thead>
<tr>
<th>Thread Races</th>
<th>Process Races</th>
</tr>
</thead>
<tbody>
<tr>
<td>27%</td>
<td>73%</td>
</tr>
</tbody>
</table>

TOCTTOU Races

23%

Thread races may be underrepresented in Linux distributions bug trackers.
General process races cannot be detected using existing race detectors.
Not so surprising

- Different programs, written in different languages
- Access many different resources
- Syscalls semantics are a bit obscure
- Depends on user configuration, specific environment
Racepro

The *first* generic process race detection framework

“They're Amazing”
Nicolas Viennot
Racepro

- Detect generic process races
- Check deployed systems in-vivo
- Low overhead
- Transparent to applications
- Detected previously known and unknown bugs
Racepro Workflow

**Deployed System**
- **Recorder**
- **Record Execution**
- **Explorer**
- **Detect Races**

**Replayed System**
- **Replayer**
- **Create Execution Branch**
- **Replay & Go-live**
- **Checker**
- **Check Failures**

- **Record** → **Detect** → **Validate**
Racepro Workflow

Deployed System

Record Execution

Record

Detect

Detect Races

Explorer

Create Execution Branch

Replay & Go-live

Replayer

Replayed System

Check Failures

Checker

Record

Detect

Validate
Racepro Workflow

Deployed System

Record Execution
Recorder

Detect Races
Explorer

Create Execution Branch

Replayed System

Replay & Go-live
Replayer

Check Failures
Checker

Record → Detect → Validate

failure
Recorder

- Builds on Scribe (Sigmetrics 2010)
- Lightweight kernel-level recorder
- Rendez-vous points:
  - Partial ordering of system calls
- Sync points:
  - Convert asynchronous events to synchronous events to track signals and shared memory
Benefits

- Tracks kernel object accesses
- Allows deterministic replay
- Enables transition to live execution
- Runs on commodity hardware, SMP friendly
- Low overhead
- Transparent to applications
ps aux | grep pizza

shell

fork

fork

ps

read(/proc/3/cmdline)

grep

extcve(grep)
Log File Content

[2] read() = 11
   read files_struct, id = 41, serial = 157
   write file, id = 152, serial = 0
   read pid, id = 40, serial = 17

[3] execve() = 0
   write pid, id = 40, serial = 8
   read inode, id = 1, serial = 0
   read inode, id = 11, serial = 0
   read inode, id = 1, serial = 0
   read inode, id = 6, serial = 0
   read inode, id = 13, serial = 0
   read inode, id = 6, serial = 0
   write futex, id = 51, serial = 0
Log File Content

[2] read() = 11
   read files_struct, id = 41, serial = 157
   write file, id = 152, serial = 0
   read pid, id = 40, serial = 17

[3] execve() = 0
   write pid, id = 40, serial = 8
   read inode, id = 1, serial = 0
   read inode, id = 11, serial = 0
   read inode, id = 1, serial = 0
   read inode, id = 6, serial = 0
   read inode, id = 13, serial = 0
   read inode, id = 6, serial = 0
   write futex, id = 51, serial = 0
Log File Content

[2] read() = 11
[2]   read files_struct, id = 41, serial = 157
[2]   write file, id = 152, serial = 0
[2]   read pid, id = 40, serial = 17

[3] execve() = 0
[3]   write pid, id = 40, serial = 8
[3]     read inode, id = 1, serial = 0
[3]     read inode, id = 11, serial = 0
[3]     read inode, id = 1, serial = 0
[3]     read inode, id = 6, serial = 0
[3]     read inode, id = 13, serial = 0
[3]     read inode, id = 6, serial = 0
[3]     write futex, id = 51, serial = 0
Log File Content

[2] read() = 11
[2] read files_struct, id = 41, serial = 157
[2] write file, id = 152, serial = 0
[2] read pid, id = 40, serial = 17

[3] execve() = 0
[3] write pid, id = 40, serial = 8
[3] read inode, id = 1, serial = 0
[3] read inode, id = 11, serial = 0
[3] read inode, id = 1, serial = 0
[3] read inode, id = 6, serial = 0
[3] read inode, id = 13, serial = 0
[3] read inode, id = 6, serial = 0
[3] write futex, id = 51, serial = 0
Log File Content

[2] read() = 11
[2]   read files_struct, id = 41, serial = 157
[2]     write file, id = 152, serial = 0
[2]     read pid, id = 40, serial = 17
[2] execve() = 0
[3]     write pid, id = 40, serial = 8
[3]       read inode, id = 1, serial = 0
[3]       read inode, id = 11, serial = 0
[3]       read inode, id = 1, serial = 0
[3]       read inode, id = 6, serial = 0
[3]       read inode, id = 13, serial = 0
[3]       read inode, id = 6, serial = 0
[3]     write futex, id = 51, serial = 0
Step 2: Detection

Log file

Races
Model

System calls are translated to \textit{load/store} micro-operations
Micro-operations

[2] read() = 11
   read files_struct, id = 41, serial = 157
   write file, id = 152, serial = 0
   read pid, id = 40, serial = 17

[3] execve() = 0
   write pid, id = 40, serial = 8
   read inode, id = 1, serial = 0
   read inode, id = 11, serial = 0
   read inode, id = 1, serial = 0
   read inode, id = 6, serial = 0
   read inode, id = 13, serial = 0
   read inode, id = 6, serial = 0
   write futex, id = 51, serial = 0
Micro-operations

[2] read files_struct, id = 41, serial = 157
[2] write file, id = 152, serial = 0
[2] read pid, id = 40, serial = 17

[3] write pid, id = 40, serial = 8
[3] read inode, id = 1, serial = 0
[3] read inode, id = 11, serial = 0
[3] read inode, id = 1, serial = 0
[3] read inode, id = 6, serial = 0
[3] read inode, id = 13, serial = 0
[3] read inode, id = 6, serial = 0
[3] write futex, id = 51, serial = 0
Micro-operations

[2] read files_struct, id = 41, serial = 157
[2] write file, id = 152, serial = 0
[3] write pid, id = 40, serial = 8
[3] read inode, id = 1, serial = 0
[3] read inode, id = 11, serial = 0
[3] read inode, id = 1, serial = 0
[3] read inode, id = 6, serial = 0
[3] read inode, id = 13, serial = 0
[3] read inode, id = 6, serial = 0
[3] write futex, id = 51, serial = 0
[2] read pid, id = 40, serial = 17
Micro-operations

[2]  load 41
[2]  store 152
[3]  store 40
[3]  load 1
[3]  load 11
[3]  load 1
[3]  load 1
[3]  load 6
[3]  load 6
[3]  store 51
[2]  load 40
Micro-operations

[2] load 41
[2] store 152
[3] store 40
[3] load 1
[3] load 11
[3] load 1
[3] load 1
[3] load 6
[3] load 6
[3] store 51
[2] load 40

You can now run your favorite thread race algorithm!
Micro-operations

You can now run your favorite thread race algorithm!
Other kinds of races...
Wait-Wakeups Race

• A waiting syscall can be woken up by many matching wakeup syscalls

• Only Racepro detect such races

• Example:
  • read() on pipe can be woken by any writers
  • waitpid() can be woken by any children
Wait-Wakeups Race Example

```
shell
  fork
  fork
  read(/proc/3/cmdline)
  execve(grep)
  wait
  wait

ps
  fork
  read(/proc/3/cmdline)
  execve(grep)
  exit
  wait
  wait

grep
  execve(grep)
  exit
```
Wait-Wakeups Race Example

- Shell
  - `fork`
  - `fork`
    - `read(/proc/3/cmdline)`
      - `execve(grep)`
        - `wait`
        - `exit`
  - `ps`
    - `wait`
    - `exit`
Step 3: Validation

Races

Harmful Races
Validation Overview

- Create execution branch: **Modified** version of the original execution that makes the race occur by changing the order of system calls
- Problem: change in the middle of the recording can make the replay diverge
- Solution: truncate the log file after the modification and transition to live execution
Validation Steps

- Deterministic replay until race occurs, including replaying internal kernel state
- Replay the reordered racy system calls
- Transition to live execution
- Run built-in or custom checkers
Is this race harmful or not?
Validation

shell

ps

grep

fork  fork  wait  wait

read(/proc/3/cmdline)

execve(grep)
Validation

shell

fork

fork

read(/proc/3/cmdline)

execve(grep)

grep

ps
Validation

Deterministic Replay
Validation

Transition to live execution
Validation

Live execution
Watched with checkers
Results

- Detected previously known and unknown bugs
- Heavy inter-process interaction
- Validation is crucial
- Recording overhead is small
## Bugs detected

<table>
<thead>
<tr>
<th>Bug</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>debian-294579</td>
<td>adduser: <code>/etc/passwd</code> corruption</td>
</tr>
<tr>
<td>debian-438076</td>
<td><code>mv</code>: unlink target before calling <code>rename</code></td>
</tr>
<tr>
<td>debian-399930</td>
<td><code>logrotate</code>: create a file that may be observed by daemons without write permissions</td>
</tr>
<tr>
<td>redhat-54127</td>
<td><code>licq</code>: <code>ps</code></td>
</tr>
<tr>
<td>launchpad-596064</td>
<td><code>upstart</code>: does not wait until <code>smbd</code> creates a directory before spawning <code>nmdb</code></td>
</tr>
<tr>
<td>launchpad-10809</td>
<td><code>bash</code>: history file corruption</td>
</tr>
<tr>
<td>new-1</td>
<td><code>tcsh</code>: history file corruption</td>
</tr>
<tr>
<td>new-2</td>
<td><code>updatedb</code>: race with <code>locate</code> when saving the database</td>
</tr>
<tr>
<td>new-3</td>
<td><code>updatedb</code>: concurrent <code>updatedb</code> may corrupt the database</td>
</tr>
<tr>
<td>new-4</td>
<td><code>abr2gbr</code>: incorrect dependencies in the <code>Makefile</code></td>
</tr>
</tbody>
</table>
# Bugs detected

<table>
<thead>
<tr>
<th>Bug</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>debian-294579</td>
<td>adduser: /etc/passwd corruption</td>
</tr>
<tr>
<td>debian-438076</td>
<td>mv: unlink target before calling rename</td>
</tr>
<tr>
<td>debian-399930</td>
<td>logrotate: create a file that may be observed by daemons without write permissions</td>
</tr>
<tr>
<td>redhat-54127</td>
<td>licq: ps</td>
</tr>
<tr>
<td>launchpad-596064</td>
<td>upstart: does not wait until smdb creates a directory before spawning nmdb</td>
</tr>
<tr>
<td>launchpad-10809</td>
<td>bash: history file corruption</td>
</tr>
<tr>
<td>new-1</td>
<td>tcsh: history file corruption</td>
</tr>
<tr>
<td>new-2</td>
<td>updatedb: race with locate when saving the database</td>
</tr>
<tr>
<td>new-3</td>
<td>updatedb: concurrent updatedb may corrupt the database</td>
</tr>
<tr>
<td>new-4</td>
<td>abr2gbr: incorrect dependencies in the Makefile</td>
</tr>
</tbody>
</table>
## Detection

<table>
<thead>
<tr>
<th>Bug</th>
<th>Processes</th>
<th>Syscalls</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>debian-294579</td>
<td>19</td>
<td>5275</td>
<td>658</td>
</tr>
<tr>
<td>debian-438076</td>
<td>21</td>
<td>1688</td>
<td>213</td>
</tr>
<tr>
<td>debian-399930</td>
<td>10</td>
<td>1536</td>
<td>279</td>
</tr>
<tr>
<td>redhat-54127</td>
<td>14</td>
<td>1298</td>
<td>229</td>
</tr>
<tr>
<td>launchpad-596064</td>
<td>34</td>
<td>5564</td>
<td>722</td>
</tr>
<tr>
<td>launchpad-10809</td>
<td>13</td>
<td>1890</td>
<td>205</td>
</tr>
<tr>
<td>new-1</td>
<td>12</td>
<td>2569</td>
<td>201</td>
</tr>
<tr>
<td>new-2</td>
<td>47</td>
<td>2621</td>
<td>467</td>
</tr>
<tr>
<td>new-3</td>
<td>30</td>
<td>4361</td>
<td>2981</td>
</tr>
<tr>
<td>new-4</td>
<td>19</td>
<td>4672</td>
<td>716</td>
</tr>
</tbody>
</table>
## Validation

<table>
<thead>
<tr>
<th>Bug</th>
<th>Detected</th>
<th>Harmful</th>
<th>Checker</th>
</tr>
</thead>
<tbody>
<tr>
<td>debian-294579</td>
<td>4231</td>
<td>42</td>
<td>Custom</td>
</tr>
<tr>
<td>debian-438076</td>
<td>50</td>
<td>4</td>
<td>Default</td>
</tr>
<tr>
<td>debian-399930</td>
<td>17</td>
<td>4</td>
<td>Default</td>
</tr>
<tr>
<td>redhat-54127</td>
<td>35</td>
<td>4</td>
<td>Custom</td>
</tr>
<tr>
<td>launchpad-596064</td>
<td>272</td>
<td>2</td>
<td>Default</td>
</tr>
<tr>
<td>launchpad-10809</td>
<td>143</td>
<td>10</td>
<td>Custom</td>
</tr>
<tr>
<td>new-1</td>
<td>137</td>
<td>14</td>
<td>Custom</td>
</tr>
<tr>
<td>new-2</td>
<td>82</td>
<td>42</td>
<td>Default</td>
</tr>
<tr>
<td>new-3</td>
<td>17</td>
<td>4</td>
<td>Default</td>
</tr>
<tr>
<td>new-4</td>
<td>8</td>
<td>1</td>
<td>Default</td>
</tr>
</tbody>
</table>
Conclusion

- Racepro: the **first** generic process race detector
  - Record applications in production systems
  - Model system calls with load/store micro-ops
  - Validate by checking uncontrolled execution

- Detected previously known and **unknown** races
- Low recording overhead
For More Information

systems.cs.columbia.edu

github.com/nviennnot/linux-2.6-scribe
## Resources

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inode</td>
<td>File, Directory, Socket, Pipe, TTY, Device</td>
</tr>
<tr>
<td>file</td>
<td>File handle of an opened file</td>
</tr>
<tr>
<td>file-table</td>
<td>Process file table</td>
</tr>
<tr>
<td>mmap</td>
<td>Process memory map</td>
</tr>
<tr>
<td>cred</td>
<td>Process credentials</td>
</tr>
<tr>
<td>global</td>
<td>System-wide properties (hostname, ...)</td>
</tr>
<tr>
<td>pid</td>
<td>Process ID</td>
</tr>
<tr>
<td>ppid</td>
<td>Parent process ID</td>
</tr>
</tbody>
</table>
Checkers

- Crash detection
- Application Hanging
- Check for error messages in log files
- Return value of application
- Linearized run (EuroSys11)