Enhanced filtering of data using data-driven analysis

Jean-Christian Kouamé
Michel Dagenais

Progress Report Meeting
May 2015
Presentation Plan

- Introduction
- Update from December
- Other features
- Advantages
- Performance
- Interesting use cases
- Filter events view
- Further work
- Demo
Introduction

• Reminder
  • Previous works (Florian Wininger)
    • Custom XML analysis
  • Goal of the project
    • Filter data
    • Detect complex defaults
    • Follow mecanism
    • Generate high-level events
Update from December

- **XML structure**
  - **3 main entities**
    - **Finite state machine (FSM)**
      - Describe the scenarios
    - **Transitions**
      - Conditions that trigger the state transitions
      - Conditions based on events or on the time
    - **Actions**
      - Action to execute
      - Supported actions:
        - State changes
        - Generate synthetic events
        - Start a new FSM
Update from December

- XML structure

```xml
<filterHandler filterName="sched_switch">
  <initialFsm id="sched_switch"/>
  <transition Input id="sched_switch">
    <event eventName="sched_switch"/>
  </transition Input>
  <action id="update Current_thread">
    <stateChange>
      <stateAttribute type="location" value="CurrentCPU"/>
      <stateAttribute type="constant" value="Current_thread"/>
      <stateValue type="eventField" value="next_tid"/>
    </stateChange>
  </action>
</fsm>
</filterHandler>
```
Others features

- Precondition
- Preaction
- Circular FSM
- FSM Singleton
- Activate/deactivate debug mode
Advantages

• Flexibility

• Isolate regions

• Realize abstraction

• Possibility of use of a graphic interface in order to design the filters (Simon's research)

• Reduce the difficulty of the analysis
Interesting use cases

- Chroot jail escape
  - Select a directory as a root folder for a file system file
  - Possibility to escape the jail with a root privilege
- Chroot(), fchdir()
Interesting use cases

- Chroot jail escape
Interesting use cases

- File access
  - Find all file access
  - From their opening to their closing
Interesting use cases

- File access
Interesting use cases

- Average of data read
  - Possibility to compute statistic
- Usage of a timer FSM
- Possibility to use XML view to show the data
Interesting use cases

• Average of data read
Interesting use cases

- Follow a process creation tree
  - Intruders could access to a server and obtain root privilege and get a shell.

- Follow all process that descending from a shell process created by Apache
Interesting use cases

- Follow process creation tree

```xml
<fsm id="fork_sh" multiple="false">
  <stateTable>
    <stateDefinition name="wait_sched_process_fork">
      <transition input="sched_process_fork" next="process_forked" action="sched_process_fork1" />
      <transition input="#other" next="wait_sched_process_fork" />
    </stateDefinition>
    <stateDefinition name="process_forked" automatic="true">
      <transition input="process_is_sh" next="wait_sched_process_fork" action="process_is_sh" />
      <transition input="#other" next="wait_sched_process_fork" />
    </stateDefinition>
    <initialState id="wait_sched_process_fork"/>
  </stateTable>
</fsm>

<fsm id="follow_process">
  <stateTable>
    <stateDefinition name="wait_sched_process_fork">
      <transition input="sched_process_fork:root_is_sh" next="wait_process_exit" action="save_data" />
      <transition input="#other" next="wait_sched_process_fork" />
    </stateDefinition>
    <stateDefinition name="wait_process_exit">
      <transition input="sched_process_exit:same_tid" next="processExited" action="gen_process_event" />
      <transition input="#other" next="wait_process_exit" />
    </stateDefinition>
    <stateDefinition name="processExited">
      <transition input="#other" next="wait_root_sh" />
    </stateDefinition>
    <initialState id="wait_sched_process_fork"/>
    <endState id="processExited" />  
  </stateTable>
</fsm>
```
Interesting use cases

- XML latency analysis
  - Get the latency data of all system calls
  - Generate a synthetic event for each system call
  - Usage of an abstraction filter of system calls
  - Work also for the IRQs
Interesting use cases

- XML latency analysis
Performance

• Criterias that affect the performance:
  • Complexity of the filter
  • Number of current scenarios

• Optimization:
  • Indexing
  • Refactoring
  • precondition
Performance (evaluation)

- Read a trace
  - Comparison between XML and Java

![Chart showing performance of reading a trace]

- Comparison of the time of reading a trace
Performance (evaluation)

- Java hard-coded filter vs XML approach
  - find all file access
  - 500MB trace file
  - 19M of events
  - 328510 relevant events
  - 1494 file access found
  - 5 coexisting FSMs

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Java</th>
<th>XML</th>
<th>% of slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>31.1155</td>
<td>32.1194</td>
<td>3.23</td>
</tr>
</tbody>
</table>
Filter events view

- A new filter tool to filter all the views
  - Based on the existing Filter View
- A user friendly view
- Possibility to manage the filters
  - Activate / Deactivate
- Filter applied on multiple views
- Updates the views instantly
Filter events view

<table>
<thead>
<tr>
<th>Type</th>
<th>Filter name</th>
<th>Enabled</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event type</td>
<td>idle</td>
<td></td>
<td>New</td>
</tr>
<tr>
<td>Event type</td>
<td>sys_open</td>
<td></td>
<td>Edit</td>
</tr>
<tr>
<td>Event type</td>
<td>sys_close</td>
<td></td>
<td>Delete</td>
</tr>
</tbody>
</table>
Filter events view
Further work

- Performance:
  - Optimize multiple scenario analysis
  - Comparison with prior work
- More use cases
  - Different fields
- The need of a “filter manager tool”
  - (Jonathan's project)
- Filter events view
  - Make more views react to the filters
• DEMO