A Runtime Environment for Online Processing of Operating System Kernel Events

Michael Schöbel, Andreas Polze

7th Intl. Workshop on Dynamic Analysis (WODA)
20. July 2009 – Chicago, IL
OS Kernel Event Tracing

- Dynamic Analysis

- Usage scenarios
  - System analysis
  - Debugging
  - Runtime-state monitoring

- Problem identification
  - Search “bad” patterns in the event stream
  - Adapt the system as reaction to “bad” pattern
Advantages of event tracing

- Detailed information
- Hints for solution might be available in trace

... under the assumption that
- Meaningful set of events is monitored
- System is usable with activated tracing
Limiting aspects

- **Problem identification**
  - Experienced administrators
  - → Pattern description

- **Huge logfiles**
  - Detailed event information requires space
  - → Online processing of events

- **Offline/Post-Mortem analysis model**
  - Activate tracing – deactivate – (reboot?) – analysis
  - → Execute callbacks / scripts when a pattern is detected
Agenda

- Pattern description
- Online processing of events
- Execute callbacks / scripts when a pattern is detected
Pattern specification

- **Similar to regular expressions**
  - Sequence \([a, b, c]\)
  - Alternative \((a \mid b \mid c)\)
  - Negation \(\sim a\)

  - Simple events \(event:name\)
  - Arrays of events \(event[>4]:name\)

  - Conditions \(WHERE\)
  - Timeframe \(WITHIN\)
  - Result data \(RETURN\)
EVENTS "wmkevents.h"

RULE nosyscall

PATTERN {
    [syscall:a, ~(syscall|threadtermination), syscall]
}

WHERE {
    [ProcessId],
    [ThreadId],
    a.SyscallNr < 300
}

RETURN {
    a.SyscallNr,
    a.TimeStamp
}
Join fields in *WHERE* statement

- **Abbreviated form ...**
  
  ```
  PATTERN {
    [syscall:a, ~(syscallexit|threadtermination), syscall]
  }
  WHERE { [ProcessId],
    [ThreadId] }
  ```

- **... instead of**

  ```
  PATTERN {
    [syscall:a, ~(syscallexit:b|threadtermination:c), syscall:d]
  }
  WHERE { a.ProcessId == b.ProcessId,
    a.ProcessId == c.ProcessId,
    a.ProcessId == d.ProcessId,
    a.ThreadId == b.ThreadId,
    a.ThreadId == c.ThreadId,
    a.ThreadId == d.ThreadId }
  ```
Based on C++ version of Coco/R

- Parse event description and pattern definition
- Generate ...
  - Deterministic Finite Automata (DFA) for pattern
  - Graphical (.dot) representation of DFA (for debugging)
  - DLL for console printing of rule results

Features

- Check \texttt{WHERE} conditions as early as possible
- Save only the required parts of the event information
- Compact binary representation of DFA
Deterministic Finite Automata

Diagram of a deterministic finite automaton with transitions and events:

- Event: EV 3 = IVAL (7) (event type)
- Transition: COPY(EV 54 -> RT.12)
- Transition: COPY(EV 10 -> RT.22)
- Transition: COPY(EV 12 -> RT.10)
- Transition: COPY(EV 16 -> RT.14)
- Transition: RT.15 < IVAL (330)
- Transition: EV 13 = RT.10 (join field)
- Transition: EV 16 = RT.14 (join field)
- Transition: EV 8 = IVAL (10) (event type)
- Transition: EV 12 = RT.10 (join field)
- Transition: EV 16 = RT.14 (join field)
- Transition: EV 8 = IVAL (7) (event type)
- Transition: EV 13 = RT.10 (join field)
- Transition: EV 16 = RT.14 (join field)
- Transition: EV 14 = RT.10 (join field)
- Transition: EV 16 = RT.14 (join field)
Agenda

- Pattern description

- Online processing of events

- Execute callbacks / scripts when a pattern is detected
Instrumentation framework

- Windows Monitoring Kernel
  - Static instrumentation
  - Based on Windows Research Kernel
    - Custom build Windows Server 2003 kernel
  - Usage similar to KLogger for Linux
  - Overhead ~ 1% for 13k events per second

- Compiler parses C header file `wmkevents.h`
  - Get available event types
  - Read event type descriptions
DFA processing model

- **Runtime state (RTS) information**
  - Representation for single automata run
  - Data structure is generated by compiler
    - Current state
    - Event field information
    - Result information
  - Evaluate conditions for current state
  - Determine valid transition
    - Conditions evaluated based on event data and RTS
    - Actions copy event data to RTS
Current implementation

■ User-mode application
  □ Load rule representation
  □ Read event stream from WMK logfile

□ Output:
  ◇ Result information
  ◇ Processing statistics

match #1
{ 106 485370171 }
mashc #2
{ 139 1320873480 }
mashc #3
{ 139 1350760491 }
mashc #4
{ 139 1351041183 }
Agenda

- Pattern description
- Online processing of events
- Execute callbacks / scripts when a pattern is detected
React to detected patterns

- **Application domain**
  - Reconfiguration
    - Caching policy
    - Number of worker threads
  - \(\rightarrow\) **Callback to application specific function**

- **System domain**
  - Reconfiguration
    - Prevent execution of malware pattern
    - Adapt thread/process priorities
  - \(\rightarrow\) **Execute script in kernel mode**
Execute application specific callback

- **Programming interface for applications**
  - Control rule lifecycle (load-activate-deactivate-unload)

- Callback
  - Registered for a specific rule
  - Implements reaction to detected pattern
  - Access to rule results (**RETURN** values)
  - Access to execution context information

- Synchronous or asynchronous processing in user mode
Kernel mode scripting

- **Extend rule specification language**
  - Allow script definition: `DO` keyword
    - Access to (named) events and event data fields
    - Access to execution context information
    - Runtime environment exposes some kernel functions
  - Execute reactions directly in the kernel
Outlook

- **Kernel integration of runtime environment**
  - Efficient synchronization
  - Condition evaluation / transition search

- **Case studies**
  - Server applications – worker thread management
  - Deadlock detection / prevention
  - Context-oriented programming
Summary

- **Pattern description**
  - Regular expressions
  - Compiled to Deterministic Finite Automata

- **Online processing of events**
  - No logfile required

- **Execute callbacks / scripts when a pattern is detected**
  - OS kernel integrated runtime environment