SystemTap

Crash Dump Analysis 2014/2015

CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

SystemTap

- Dynamic analysis framework
- Inspired by DTrace
- Linux, GPL
- Under heavy development
- Suitable for production usage

Motto: *“Painful to use, but more painful not to”*

Tracing

- Capture information about execution
- Often specialized tools (strace)
- Limited filtering
- Overwhelming amount of data

Profiling

- Sampling during execution
- Both targeted and system-wide
- Often just one metric: time

Debugging

- Full context (memory, registers, backtrace)
- Breakpoints
- Targeted

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

Environments

- Development
  - Rich environment: tools, debuginfo
  - Exclusive access and control
- Production
  - Different from development
  - Under load
  - Valuable
  - Virtualized / containerized

SystemTap to rule them all, in production

- Tracing, profiling, debugging
- System-wide
- Monitoring multiple event types (and layers)
- Safe
- Low overhead
- Controlled dependencies

Basic principles

- Events
  - Interesting points in running system
  - Different abstraction levels: code line/model event
  - Kernel and user space
- Actions
  - Scripting language
  - Access to program state, capturing, processing
  - Even state modification if you are bold

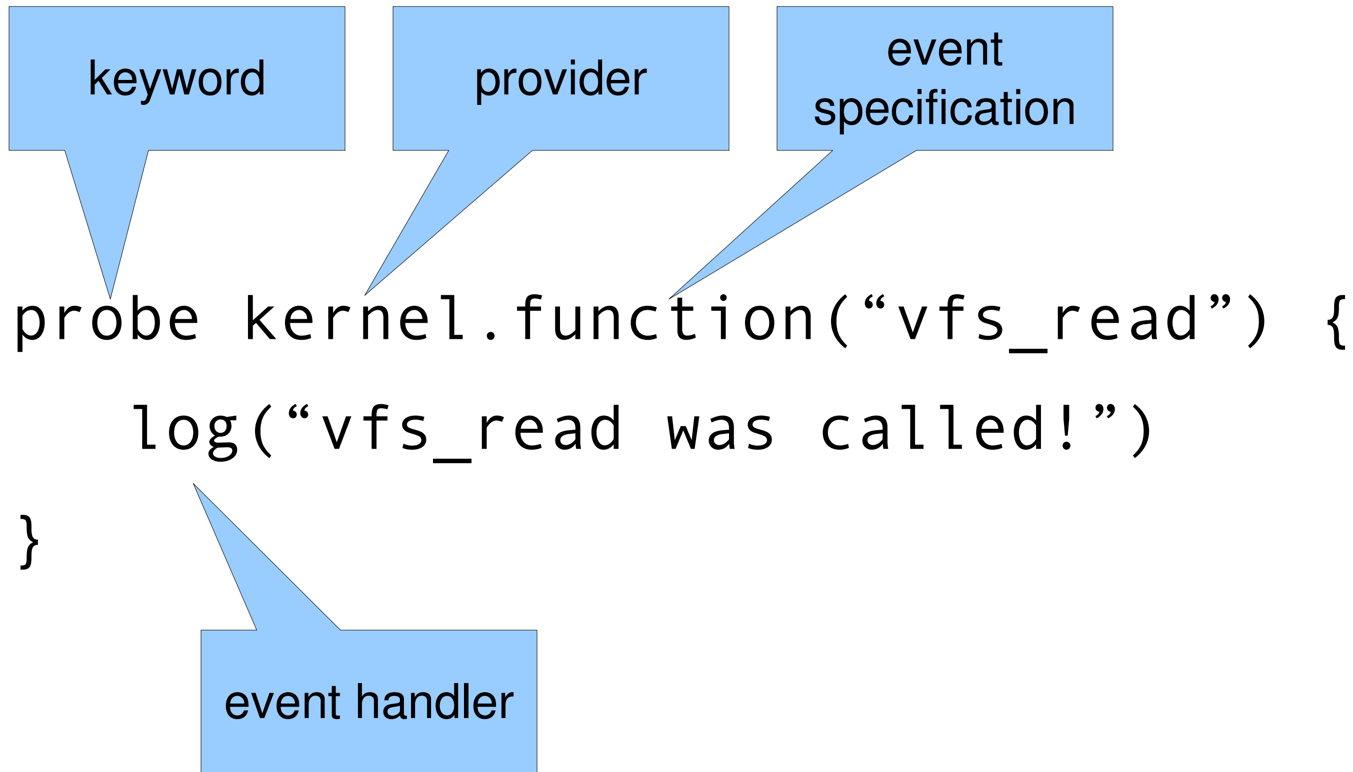
Events

- Low-level
  - Function entry and return
  - Code lines
- High-level
  - Timers
  - Code tracepoints
  - Tapsets

Events: Probes

- **Probe:** Basic SystemTap element
  - Bind actions to events
- **Probe specification**: Defines interesting events
- **Probe hit:** Specific event occurrence
- **Probe handler:** Action to perform when hit

Language: Probes



Language: Probes

Probes

Probes: Generic

- begin
- end
- error
- timer.jiffies()
- timer.ms()

Probes: System calls

- Needs DWARF debuginfo
  - syscall.NAME
  - syscall.NAME.return
- Do not need DWARF debuginfo
  - nd_syscall.NAME
  - nd_syscall.NAME.return

Probes: Kernel DWARF

- kernel.function(PATTERN)
- module(PATTERN).function(PATTERN)
- kernel.statement(PATTERN)
- module(PATTERN).statement(PATTERN)

- Provides access to context variables, arguments, reachable memory etc.

Probes: Modifiers and variants

- function.return
- function.call / function.inline
- function.callee(NAME)
- function.callees(DEPTH)

Probes: Kernel DWARF-less

- kprobe.function(FUN) + .return
- kprobe.module(MOD).function(FUN) + .return

- No access to variables etc.

Probes: Kernel tracepoints

- kernel.trace(PATTERN)

- Static markers in kernel
- Faster and more reliable mechanism
- Access to macro arguments

Probes: Userspace DWARF

- process(PATH/PID).function(NAME)
- process(PATH/PID).statement(PATTERN)
- process(P).library(P).function(NAME)
- process(P).library(P).statement(PATTERN)

- Provides access to context variables, arguments, reachable memory etc.

Probes: Userspace via utrace

- process({,PATH,PID}).{begin,end}
- process({,PATH,PID}).thread.{begin,end}
- process({,PATH,PID}).syscall + .return

Probes: Userspace static markers

- process(PID/PATH).mark(LABEL)
- process(P).provider(PROVIDER).mark(LABEL)

- Static markers put by developers to application
- STAP_PROBE/DTRACE_PROBE macros
- Access to macro arguments

Probes: Python

- Markers in Python runtime
- Tapset to work with Python structure

Probes: Java

- Prototype feature
- Byteman backend

Example:

```
probe java(PNAME/PID).class(NAME).method(METHOD)
```

Probes: Find available probes

- stap -l 'probe definition'

Example:

```
$ stap -l 'kernel.trace(“kmem:*”)'
```

Actions

Actions

Language: Elements [1/2]

- **Functions:** built-in, user defined, tapsets
- **Probe aliases**
- **Control structures:** conditionals, loops, ternary op
- **Variables**
  - Scalar: string, integers, no declarations
  - Multi-key associative arrays (global only)
  - Global or local scope
- **Aggregates**

Language: Elements [2/2]

- **Operators**: Usual ops, member operator
- Pointer typecasting
- Conditional compilation
- Simple preprocessor macros
- **Embedded C**

Language: Reading

- **Context and tapset functions**
  - pid(), execname(), print_backtrace()
- **Context variables**
  - Prefixed by '$'
  - $ stap -L 'probe kernel.DEF'
- **Tapset variables**
  - Set by a tapset in a prologue
- **Pretty printers and groups**
  - $$vars, $$locals, $$parms, $$parms$, $$parms$$

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

Language: Parameters

- $ stap script.stp param1 param2 ...
- **Unquoted:** $1, $2...
- **Strings:** @1, @2...

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

Language: Output

- **Functions**
  - log()
  - printf()
  - sprintf()
  - ...more...

Language: Aggregates

- **Aggregates:** Support for data accumulation and statistics
  - aggregate <<< value
  - array[execname(), pid()] <<< value
  - @count, @sum, @min, @max, @avg
  - @hist_linear(agg, LOW, HIGH, WIDTH)

Language: Quirks

- Often quirky syntactic shortcuts
- @entry(expression): saves expression value in entry probe for usage in return probe
- foreach(v = [i,j] in array)

- Read language reference...

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

Tapsets

- **Tapset** = “Library”
- Collection of SystemTap functions and aliases
- Encapsulates internals via aliases
- Provides convenience functions

Example:

/usr/share/systemtap/tapset/linux/syscalls2.stp

Usage

Using SystemTap

- Write a script
- One-shot script execution
- Long-term result collection
- Flight-recorder mode

SystemTap passes

- **Pass 1:** Parser
- **Pass 2:** Process probes, data structures etc.
- **Pass 3:** Translate to C (cached)
- **Pass 4:** Compile as a kernel module (cached)

- **Pass 5:** Insert module

SystemTap translator / driver

- **Command:** stap
- High-level driver command
- All or some passes
- System-wide or targeted

Examples:

```
$ stap -e 'probe kernel.trace(“kmem:*”)
$ stap script.stp
$ stap -p4 script.stp
$ stap (…) -c 'command'
$ stap (…) -x PID
```

SystemTap runtime

- **Command:** staprun
- Loads a SystemTap probe kernel module
- stap = stap -p4 + staprun

Examples:

```
$ staprun stap_ee1d4debf0add5c64f0b095_1991.ko
```

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

Permission model

- Privileged operations
- Therefore, SystemTap needs either:
  - root user
  - user in groups: stapusr+stapdev
  - user in groups: stapusr+stapsys
  - user in groups: stapusr

Permission model

- stapusr + stapdev
  - Can run SystemTap as if root
- stapusr + stapsys
  - Can run prebuilt, signed modules
  - Can run limited functionality scripts
    - Avoid damage to system
- stapusr
  - Can run prebuilt and/or signed modules
  - Can run limited functionality scripts
    - Disallow access to other user information
    - Disallow harming other user process performance

Flight recorder and SystemTap service

- **Flight recorder**
  - $ stap -F ...
  - Load module and detach...
  - ...or run staprun as a daemon
- **SystemTap service**
  - /etc/systemtap/script.d/script.stp
  - /etc/systemtap/conf.d/script.conf
  - `$ service systemtap start script`

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

Guru mode

- **Fun, useful and a little bit perilous!**
- Bypass security limits
- Embedded C
- Write to context variables
- Usage: hotfixes, hacks, showcases, fun

Example:

```
$ stap -g -e 'probe syscall.kill {if (pid==$1)
{ $pid = pid() } }' PID
```

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

SystemTap compile server

- **Server**
  - Has SystemTap translator and module builder
  - Has DWARF debuginfo
  - Builds and signs SystemTap modules
- **Client**
  - Have just SystemTap client and runtime

Final words

Final words

SystemTap: Problem solving process

- Determine involved components
  - Kernel, executables, libraries
- Investigate possible probe points
  - What events are interesting?
  - What events carry interesting data?
- Determine desired output
  - Histogram, specific knowledge, data set
- Start with generic probes, then limit

SystemTap: Patterns

- Event logger: simple output on event
- Event detector: complicated filtering in probes
- Top: data collection, printing in timer.ms
- Data collection: runs until cancel, end probe

SystemTap: Resources

- man pages
  - stap, stapprobes
- Examples
  - https://sourceware.org/systemtap/examples/
- Tutorials and articles
  - https://sourceware.org/systemtap/wiki
- Language, tapset references
  - https://sourceware.org/systemtap/documentation.html