LTTng-UST dynamic tracepoints in uftrace

Progress Report Meeting

Clément Guidi Mohammad Nassiri

DORSAL – Polytechnique Montréal

January 14, 2022

Table of contents

1. Introduction
2. Previous achievements
   - Main contributions
   - Side improvements
3. LTTng-UST tracepoints
   - Tracepoint definition
   - Using uftrace features
   - Visualization
   - Difficulties encountered
   - Demo
4. Work in progress and future research
   - Work in progress
   - Future work
5. Conclusion

Introduction

About uftrace:

- function tracing tool for C/C++/Rust applications
- can instrument userspace
- empowering features, including
  - plain and regex filters for function/library names
  - execution duration and call depth filters
  - argument and return value logging

Upstream limitations:

- binary instrumentation performed before execution
- custom trace format

Previous achievements

Main contributions

Previous work at DORSAL includes:

- indirect jump resolution: improve patching success rate by identifying **indirect jump locations** (external library) *[Gabriel Pollo-Guilbert]*
- x86 runtime patching and unpatching: instrument binaries during execution using a **locking mechanism** and **out of line execution**; remove tracepoints *[Christian Harper-Cyr, Anas Balboul, Ahmad Shahnejat and Gabriel Pollo-Guilbert]*
- client command: send commands to a **libmcount daemon** running inside a uftrace target *[Clément Guidi]*

Previous achievements

Side improvements

Smaller improvements have been made:

- read external symbol file for stripped binaries – using `--with-syms=DIR` option
- detailed patching statistics
- unpatch option enhancement, for the new unpatching capabilities
- bug fixes
  - Intel CET ENDBRANCH instruction was sometimes omitted
  - cache serialization: membarrier `MEMBARRIER_CMD_PRIVATE_EXPEDITED_SYNC_CORE` command unavailable on older kernels; use CPUID interrupt instead

Previous achievements

Side improvements

Example of detailed statistics in debug mode when instrumenting `python3.11`.

```
dynamic: dynamic patch stats for 'python3.11'
dynamic:    total:     1479
dynamic:  patched:      602 (40.70%)
dynamic:   failed:      853 (57.67%)
dynamic:     no detail:        0 (  0.00%)
dynamic: relative jump:       64 ( 72.72%)
dynamic: relative call:        0 (  0.00%)
dynamic:           PIC:       24 ( 27.27%)
dynamic:  skipped:       24 ( 1.62%)
dynamic: no match:        0
```

LTTng-UST tracepoints

uftrace makes use of:

- custom trace format – serialized timestamped events
- custom event buffering mechanism

Objective Leverage uftrace capabilities to produce LTTng CTF traces

- filters
- dynamic patching
- argument and return value logging

Solution Emitting LTTng events in regular libmcount probe entries.

LTTng-UST tracepoints

Tracepoint definition

```
#define TRACEPOINT_PROVIDER lttng_ust_cyg_profile

#include <lttng/tracepoint.h>

TRACEPOINT_EVENT_CLASS(
  lttng_ust_cyg_profile,
  func_class,
  TP_ARGS(
    void *, func_addr,
    void *, call_site,
    char *, arg_ret_str),
  TP_FIELDS(
    ctf_integer_hex(unsigned long, addr, (unsigned long) func_addr)
    ctf_integer_hex(unsigned long, call_site, (unsigned long) call_site)
    ctf_string(arg_ret_str, arg_ret_str)
  )
)
```

LTTng-UST tracepoints

Tracepoint definition

```
TRACEPOINT_EVENT_INSTANCE(
  lttng_ust_cyg_profile,
  func_class,
  func_entry,
  TP_ARGS(void *, func_addr,
            void *, call_site,
            char *, arg_ret_str)
)

TRACEPOINT_EVENT_INSTANCE(
  lttng_ust_cyg_profile,
  func_class,
  func_exit,
  TP_ARGS(void *, func_addr,
            void *, call_site,
            char *, arg_ret_str)
)
```

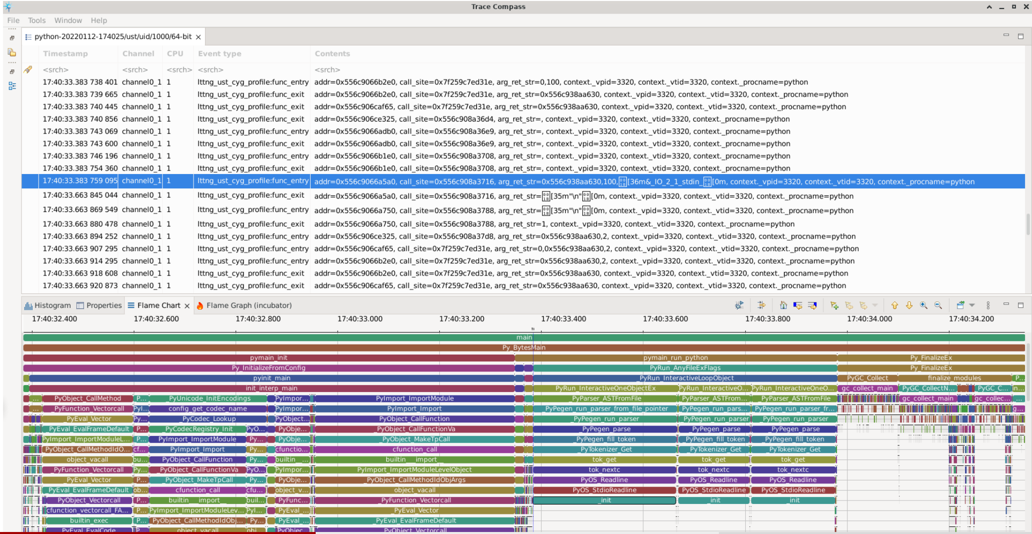
LTTng-UST tracepoints

Using uftrace features

- Function/library filter: works normally, handled by libmcount at function entry
- Call depth filter: works normally, handled by libmcount at function entry
- Time filter: **not** functional, handled at function exit. Entry event still emitted
- Argument and return value: functional, available in `arg_ret_str` event field

LTTng-UST tracepoints

Visualization



Timestamp	Channel	CPU	Event type	Contents
<srch>	<srch>	<srch>	<srch>	<srch>
17:40:33.383 738 401	channel0_1	1	lttng_ust_cyg_profile:func_entry	addr=0x556c9066b2e0, call_site=0x7f259c7ed31e, arg_ret_str=0,100, context._vpid=3320, context._vtid=3320, context._procname=python
17:40:33.383 739 665	channel0_1	1	lttng_ust_cyg_profile:func_exit	addr=0x556c9066b2e0, call_site=0x7f259c7ed31e, arg_ret_str=0x556c938aa630, context._vpid=3320, context._vtid=3320, context._procname=python
17:40:33.383 740 445	channel0_1	1	lttng_ust_cyg_profile:func_exit	addr=0x556c906caf65, call_site=0x7f259c7ed31e, arg_ret_str=0x556c938aa630, context._vpid=3320, context._vtid=3320, context._procname=python
17:40:33.383 740 856	channel0_1	1	lttng_ust_cyg_profile:func_exit	addr=0x556c906ce325, call_site=0x556c908a36d4, arg_ret_str=, context._vpid=3320, context._vtid=3320, context._procname=python
17:40:33.383 743 069	channel0_1	1	lttng_ust_cyg_profile:func_entry	addr=0x556c9066adb0, call_site=0x556c908a36e9, arg_ret_str=, context._vpid=3320, context._vtid=3320, context._procname=python
17:40:33.383 743 600	channel0_1	1	lttng_ust_cyg_profile:func_exit	addr=0x556c9066adb0, call_site=0x556c908a36e9, arg_ret_str=, context._vpid=3320, context._vtid=3320, context._procname=python
17:40:33.383 746 196	channel0_1	1	lttng_ust_cyg_profile:func_entry	addr=0x556c9066b1e0, call_site=0x556c908a3708, arg_ret_str=, context._vpid=3320, context._vtid=3320, context._procname=python
17:40:33.383 754 360	channel0_1	1	lttng_ust_cyg_profile:func_exit	addr=0x556c9066b1e0, call_site=0x556c908a3708, arg_ret_str=, context._vpid=3320, context._vtid=3320, context._procname=python
17:40:33.383 759 095	channel0_1	1	lttng_ust_cyg_profile:func_entry	addr=0x556c9066a5a0, call_site=0x556c908a3716, arg_ret_str=0x556c938aa630,100,␛[36m&_IO_2_1_stdin_␛[0m, context._vpid=3320, context._vtid=3320, context._procname=python
17:40:33.663 845 044	channel0_1	1	lttng_ust_cyg_profile:func_exit	addr=0x556c9066a5a0, call_site=0x556c908a3716, arg_ret_str=␛[35m"\n"␛[0m, context._vpid=3320, context._vtid=3320, context._procname=python
17:40:33.663 869 549	channel0_1	1	lttng_ust_cyg_profile:func_entry	addr=0x556c9066a750, call_site=0x556c908a3788, arg_ret_str=␛[35m"\n"␛[0m, context._vpid=3320, context._vtid=3320, context._procname=python
17:40:33.663 880 478	channel0_1	1	lttng_ust_cyg_profile:func_exit	addr=0x556c9066a750, call_site=0x556c908a3788, arg_ret_str=1, context._vpid=3320, context._vtid=3320, context._procname=python
17:40:33.663 894 252	channel0_1	1	lttng_ust_cyg_profile:func_entry	addr=0x556c906ce325, call_site=0x556c908a37d8, arg_ret_str=0x556c938aa630,2, context._vpid=3320, context._vtid=3320, context._procname=python
17:40:33.663 907 295	channel0_1	1	lttng_ust_cyg_profile:func_entry	addr=0x556c906caf65, call_site=0x7f259c7ed31e, arg_ret_str=0,0x556c938aa630,2, context._vpid=3320, context._vtid=3320, context._procname=python
17:40:33.663 914 295	channel0_1	1	lttng_ust_cyg_profile:func_entry	addr=0x556c9066b2e0, call_site=0x7f259c7ed31e, arg_ret_str=0x556c938aa630,2, context._vpid=3320, context._vtid=3320, context._procname=python
17:40:33.663 918 608	channel0_1	1	lttng_ust_cyg_profile:func_exit	addr=0x556c9066b2e0, call_site=0x7f259c7ed31e, arg_ret_str=0x556c938aa630, context._vpid=3320, context._vtid=3320, context._procname=python
17:40:33.663 920 873	channel0_1	1	lttng_ust_cyg_profile:func_exit	addr=0x556c906caf65, call_site=0x7f259c7ed31e, arg_ret_str=0x556c938aa630, context._vpid=3320, context._vtid=3320, context._procname=python

LTTng-UST tracepoints

Difficulties encountered

- Preserving registers: mcount saves the registers it alters before executing the probe. LTTng tracepoints are using previously untouched vector registers `ymm0` and `ymm1`. mcount needs to save/restore them.
- Time filter: the time filter mechanism is incompatible with LTTng, as function entry events would need to be sent conditionally at the end of the function. This alters the timestamp.
  - Hints: only log exit event with the duration, keeping a stack of conditional events
- Upstream refactoring: conflicting refactoring was applied upstream.

LTTng-UST tracepoints

Demo

How to use uftrace with LTTng:

1. LTTng session: add `vpid`, `vtid` and `procname` userspace context

   ```
   lttng create my-session
   lttng enable-event -u -a # all userspace events
   lttng add-context -u -t vpid -t vtid -t procname
   lttng start
   ```

2. uftrace: instruct uftrace to use `libmcount-lttng.so` library using `--libmcount-lttng` option
3. instrumentation: instrumenting the target is not mandatory, it can be done at runtime. Use `--dynamic` to initialize the relevant mechanism
4. runtime: send patching/unpatching instruction with the client, using regular `--patch`/`-P` and `--unpatch`/`-U` options

LTTng-UST tracepoints

Demo

Work in progress and future research

Work in progress

Evaluating the performance of dynamic binary instrumentation according to the following criteria:

- patching success rate: the percentage of locations that are successfully instrumented
- patching perturbation: the time needed to instrument functions or remove instrumentation, and global slowdown caused to the target
- probe overhead: slowdown caused by the execution of probes
- memory consumption

We will evaluate performance on a list of around 30 applications with the following characteristics:

- C, C++ or Rust language
- low or high function count
- small or big binary size
- single- or multi-threaded

Work in progress and future research

Future work

Next project steps:

- improve instrumentation methods to increase patching success rate and efficiency
  - use 2-byte relative jumps with intermediate trampolines
  - use instruction punning
- support ARM platforms
- validate the robustness
- attach on the fly to running process
  - a PR exists on GitHub but is on hold
- apply methods to other tools: Kprobes, GDB
- support adaptative tracing: continuously patch and unpatch function based on usage

Conclusion

Source code repository: `https://gitlab.com/dorsal1/uftrace`

Questions?