Tracy - UNIX system call tracing

Merlijn Wajer

August 2, 2013

Table of contents

Introduction

Motivation:

- Cross architecture, cross platform system call tracing
- ptrace is not cross architecture

Introduction

Motivation:

- Cross architecture, cross platform system call tracing
- ptrace is not cross architecture

Possible use cases:

- Jails per process(group)
- Transparent routing of I/O
- Debugging (visualisation, replaying)
- Fault injection
- I/O logging

Background

System call:

- Fundamental interface between an application and the (Linux) kernel
- Interaction with hardware, other processes
- Invoked with a system call instruction or interrupt

Examples: **open**, **write**, **socket**, **fork**, **wait**

Background II

ptrace(2):

- System call: process trace
- Observing and controlling the execution of another process
  - Trap on every instruction; or
  - Trap on syscalls and signals
- Not **POSIX**
- No uniform API

Background II

ptrace(2):

- System call: process trace
- Observing and controlling the execution of another process
  - Trap on every instruction; or
  - Trap on syscalls and signals
- Not **POSIX**
- No uniform API

Tracy does make use of ptrace on every platform that Tracy supports

Background III

ptrace(2) allows Tracy to:

- Trap on system call instructions
- Modify registers and memory
- Control the signals sent to the program
- Spawn new processes that are traced immediately or to attach to running processes

Features of Tracy

- Event based (syscalls and signals)

Features of Tracy

- Event based (syscalls and signals)
- Syscalls, signals can be hooked, as opposed to getting an event for everything

```
tracy_set_hook(tracy, "read", TRACY_ABI_NATIVE,
    &read_hook);
```

Features of Tracy

- Event based (syscalls and signals)
- Syscalls, signals can be hooked, as opposed to getting an event for everything

```
tracy_set_hook(tracy, "read", TRACY_ABI_NATIVE,
    &read_hook);
```

- System call injection:

```
tracy_inject_syscall(child, __NR_getpid, NULL, &pid);
```

Features of Tracy

- Event based (syscalls and signals)
- Syscalls, signals can be hooked, as opposed to getting an event for everything

```
tracy_set_hook(tracy, "read", TRACY_ABI_NATIVE,
    &read_hook);
```

- System call injection:

```
tracy_inject_syscall(child, __NR_getpid, NULL, &pid);
```

- (Fast) memory access to process being traced

```
tracy_read_mem(child, dest, src, sizeof(char) * 10);
```

Features of Tracy

- Event based (syscalls and signals)
- Syscalls, signals can be hooked, as opposed to getting an event for everything

```
tracy_set_hook(tracy, "read", TRACY_ABI_NATIVE,
    &read_hook);
```

- System call injection:

```
tracy_inject_syscall(child, __NR_getpid, NULL, &pid);
```

- (Fast) memory access to process being traced

```
tracy_read_mem(child, dest, src, sizeof(char) * 10);
```

- Support for x86, AMD64 and ARM.
- Experimental bindings for Python

Jargon

- tracee: program being traced
- tracer: program tracing another program

Jargon

- tracee: program being traced
- tracer: program tracing another program

Tracy is not the “tracee”!

Implementation: injection

ptrace stops tracee before and after system call, main idea:

- When program performs a system call
- Replace system call number and arguments
- After completion: “restore” to previous state, including ip

Implementation: injection

ptrace stops tracee before and after system call, main idea:

- When program performs a system call
- Replace system call number and arguments
- After completion: "restore" to previous state, including ip

Distinguish between "sync" and "async" injection:

- Synchronous injection blocks
- Asynchronous injection returns immediately, generates event at a later time

Implementation: pre-injection

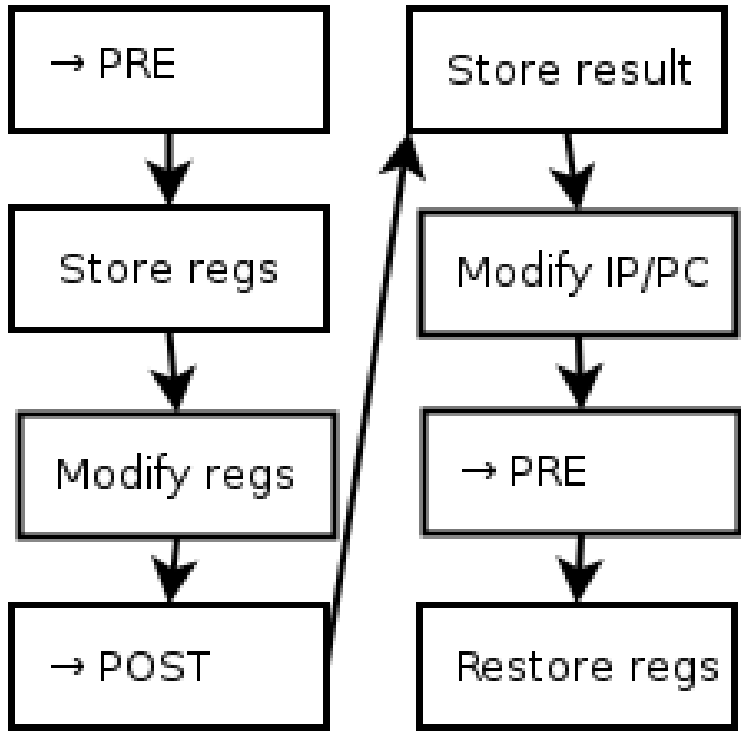

Figure : Injection from pre-system call

Implementation: memory access

Memory access using:

- ptrace (*POKETEXT*, *PEEKTEXT*)

Implementation: memory access

Memory access using:

- ptrace (*POKETEXT*, *PEEKTEXT*)
  Slow!

Implementation: memory access

Memory access using:

- ptrace (*POKETEXT*, *PEEKTEXT*)
  Slow!
- */proc/PID/mem*

Implementation: memory access

Memory access using:

- ptrace (*POKETEXT*, *PEEKTEXT*)
  Slow!
- */proc/PID/mem*
  Requires kernel $\geq$ 2.6.39

Implementation: memory access

Memory access using:

- ptrace (*POKETEXT*, *PEEKTEXT*)
  Slow!
- */proc/PID/mem*
  Requires kernel $\geq$ 2.6.39
- *process_vm_readv* and *process_vm_writev*

Implementation: memory access

Memory access using:

- ptrace (*POKETEXT*, *PEEKTEXT*)
  Slow!
- */proc/PID/mem*
  Requires kernel $\geq$ 2.6.39
- *process_vm_readv* and *process_vm_writev*
  Requires kernel $\geq$ 3.6

Memory sharing

Race condition: memory (pointed to by syscall arg) may be modified after “verification”

Memory sharing

Race condition: memory (pointed to by syscall arg) may be modified after "verification"

Solution:

- Copy over to memory which tracee can only read from
- Change syscall arguments
- Read-Only for tracee
- Read-Write for tracer

Safe-fork

Problem: Tracing children created with **fork**

Safe-fork

Problem: Tracing children created with **fork**

Linux has feature to automagically trace created children; other platforms do not

Safe-fork

Problem: Tracing children created with **fork**

Linux has feature to automagically trace created children; other platforms do not

Solution: Run syscalls like **fork** in a controlled environment: spin child execution until we have the pid of the child.

Trampy (safe-fork)

Inject code:

```
fork();
send_pid();
while (1) {
    sched_yield();
}
```

Multiple ABIs

Linux allows processes to use multiple ABIs...

Very messy:

- Instruction determines ABI
- ... but the cs register also has an effect
- Processes can mix ABIs at runtime

Strace does it wrong

strace?

- 25000 lines of code
- No architecture specific files
- In other words: death by ifdef
- *get_scno* function is 450 lines long, all platform-specific code inlined
- And... it doesn't work (well)

strace gone wrong

64 bit program:

```
__asm__(
    "int $0x80"
    :
        "=a"(pid)
    :
        "a"(20)
    );
```

strace will see “writev”

strace gone wrong

64 bit program:

```
__asm__(
    "int $0x80"
    :
        "=a"(pid)
    :
        "a"(20)
    );
```

strace will see “writev”

Tracy does it properly

strace cont.

```
...
writev(1, [{"", 0}, {..., 140736580874188}, {"", 0},
    {process_vm_readv: Bad address
0x21, 140736581373952}, {process_vm_readv: Bad address
0x10, 395049983}, {process_vm_readv: Bad address
...
```

tracy output

```
...
1688 System call: getpid (20) Pre: 1
1688 System call: getpid (20) Pre: 0
...
```

tracy output

```
...
1688 System call: getpid (20) Pre: 1
1688 System call: getpid (20) Pre: 0
...
```

Clue:

```
$ ./syscall 20
i386        getpid
x86_64      writev
```

Application: Soxy

- Transparent proxifier using SOCKS 5.
- Like tsocks (or torsocks)

Application: Soxy

- Transparent proxifier using SOCKS 5.
- Like tsocks (or torsocks)

But does not use LD_PRELOAD; so it will actually catch every network system call.

Application: Soxy

- Transparent proxifier using SOCKS 5.
- Like tsocks (or torsocks)

But does not use LD_PRELOAD; so it will actually catch every network system call.

However, currently Soxy is buggy when it comes to supporting all the ABIs.

LD_PRELOAD

Not an alternative; although may be more favourable in some cases:

- Generally easier to use
- Depends on a certain library being loaded, does not work if programs directly invoke system calls

Speed: kernel patch

Overhead: Context switches, performing system calls

Example:

```
r = ptrace(PTRACE_SETSYSCALLMASK, pid, 1, __NR_read);
r = ptrace(PTRACE_SETSYSCALLMASK, pid, 1, __NR_write);
r = ptrace(PTRACE_SYSCALLWHITELIST, pid, NULL, 0);
if (r) {
    fprintf(stderr, "New API failed... :-( .\n");
    kill(pid, SIGKILL);
    waitpid(pid, &status, 0);
    return NULL;
}
```

Future applications

Possibilities are endless:

- Buggy: Fault injection for application testing
- Jelly: Secure jail in userspace
- Fussy: FUSE in User Space (fake /dev/fuse)
- stracy: Proper strace

Future work

- **Threadsafe** ABI detection
- Research: How cross platform (or arch) can we become?
- Safely tracing children on *BSD: **Safe-fork**
- Speed: working on a proper kernel patch
- System call Intermediate Representation

Resources for Tracy

- Authors:
  - Merlijn Wajer (Wizzup)
  - Bas Weelinck (meridion)
  - Jurriaan Bremer (skier_)
- Idea: Ilja Kamps (Ikarus)
- Source, Documentation:
  `https://github.com/MerlijnWajer/tracy`
- IRC: #tracy on Freenode
- http://hetgrotebos.org/wiki/Tracy
- ML: `tracy@freelists.org`