strace: fight for performance

Eugene Syromiatnikov, Dmitry Levin

FOSDEM 2020

What is `strace`?

`strace` is a diagnostic, debugging and instructional userspace utility for Linux. It is used to monitor and tamper with interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state.

— *https://strace.io/*

What is `strace`?

`strace` is a diagnostic, debugging and instructional userspace utility for Linux. It is used to monitor and tamper with interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state.

— *`https://strace.io/`*

BUGS
A traced process runs slowly.

— *`strace(1)`*

Why is `strace` slow? [1/2]

Why is strace slow?

ptrace(2)

- strace utilizes ptrace infrastructure for tracing
- ptrace(2) is a generic debugging interface that provides a set of commands (*requests*) that enable various operations: reading and writing tracee's memory, obtaining tracee's registers, and so on
- Almost all ptrace operations are performed on a stopped process
- The ptrace API (ab)uses the standard UNIX parent/child signaling over waitpid(2) in order to deliver notifications about changes in tracees' state (including ptrace-induced stops)
- This mechanism is used to notify tracer about all kinds of events: syscall stops (after resume with PTRACE_SYSCALL request), signal deliveries, group stops, forks, execs, etc.

Why is `strace` slow?

How `strace` traces processes

- `strace` is waiting for events in `wait4(2)`
- Upon receiving a `ptrace` event, `strace` tries to figure out what happened (syscall stop, signal received by tracee, etc.)
    - The only information it has at this point is the status returned by the `wait4(2)` syscall; as a result, additional `PTRACE_GETEVENTMSG` request is required to distinguish some of the events
- For syscall stops, the additional information (syscall number and arguments on entering, return code on exiting) is retrieved
- Syscall-specific decoder function is called, which, in turn, may perform additional reads from tracee's memory for elaborate argument printing (structures, arrays, linked lists...)
- When the decoding is finished, tracee is resumed
- For each syscall, syscall stop is happened twice: on syscall entering and exiting

Syscall argument retrieval

How syscall arguments are obtained

- `ptrace` provides only basic requests for reading registers[1]
- Different architectures provide different means for it:
  - `PTRACE_PEEKUSER`
  - `PTRACE_GETREGS`
  - `PTRACE_GETREGSET NT_PRSTATUS`

[1] Unless `PTRACE_GET_SYSCALL_INFO` request is available

Syscall argument retrieval

How syscall arguments are obtained

- ptrace provides only basic requests for reading registers[1]
- Different architectures provide different means for it:
  - PTRACE_PEEKUSER
  - PTRACE_GETREGS
  - PTRACE_GETREGSET NT_PRSTATUS

Use PTRACE_GETREGSET whenever possible

```
commit eec8d5d6b028665a73169fda96e4e873cb8351f0
Author: Denys Vlasenko <vda.linux@googlemail.com>
Date:   Thu Feb 14 03:29:48 2013 +0100

    [X86] Use ptrace(PTRACE_GETREGSET, NT_PRSTATUS) to get registers.

    Unlike PTRACE_GETREGS, this new method detects 32-bit processes
    reliably, without checking segment register values which
    are undocumented and aren't part of any sort of API.
    While at it, also fixed x32 detection to use __X32_SYSCALL_BIT,
    as it should have been from the beginning.
```

[1] Unless PTRACE_GET_SYSCALL_INFO request is available

Syscall argument retrieval

Without PTRACE_GETREGSET

```
ptrace(PTRACE_PEEKUSER, 7486, 8*ORIG_RAX, [0x2]) = 0
ptrace(PTRACE_PEEKUSER, 7486, 8*CS, [0x33]) = 0
ptrace(PTRACE_PEEKUSER, 7486, 8*RAX, [0xffffffffffffffda]) = 0
ptrace(PTRACE_PEEKUSER, 7486, 8*RDI, [0x7f8cb6d24640]) = 0
ptrace(PTRACE_PEEKUSER, 7486, 8*RSI, [0x80000]) = 0
ptrace(PTRACE_PEEKUSER, 7486, 8*RDX, [0x7f8cb6d27150]) = 0
```

With PTRACE_GETREGSET

```
ptrace(PTRACE_GETREGSET, 7510, 0x1, 0x6f0440) = 0
```

Syscall argument retrieval

Do not read syscall number on exiting

```
commit 77a7459536f38dd35364c24719ce5ca5cd6b76bc
Author: Denys Vlasenko <dvlasenk@redhat.com>
Date:   Wed Aug 24 16:56:03 2011 +0200

    Do not read syscall no in get_scno_on_sysexit
```

Do not call ptrace for filtered syscalls on exiting

```
commit 7df7bc1889ca2d75341ff4e4f7ee5e5903bf7b88
Author: Dmitry V. Levin <ldv@altlinux.org>
Date:   Tue Apr 11 04:04:37 2017 +0000

    trace_syscall_exiting: do not call get_regs for filtered syscalls

    This saves up to 25% of ptrace syscalls in case of trace filtering.
```

Syscall argument retrieval

Avoid unnecessary ptrace calls

```
commit ce7d953ebecc10f71e191b6d18cfeb2399429d5f
Author: Denys Vlasenko <vda.linux@googlemail.com>
Date:   Tue Feb 5 16:36:13 2013 +0100

    Optimize out PTRACE_PEEKUSER with -i

    strace -i was fetching PC with a separate PEEKUSER
    despite having GETREGS data:

    ptrace(PTRACE_GETREGS, 22331, 0, 0x8087f00) = 0
    ptrace(PTRACE_PEEKUSER, 22331, 4*EIP, [0x80dd7b7]) = 0
    write(3, "[080dd7b7] ioctl(0, SNDCTL_TMR_T"..., 82) = 82
    ptrace(PTRACE_SYSCALL, 22331, 0, SIG_0) = 0

    Now it does this:

    ptrace(PTRACE_GETREGS, 22549, 0, 0x8087ea0) = 0
    write(3, "[080dd7b7] ioctl(0, SNDCTL_TMR_T"..., 82) = 82
    ptrace(PTRACE_SYSCALL, 22549, 0, SIG_0) = 0

    Analogous improvement in sys_sigreturn() is also implemented.
```

Syscall data retrieval

Use process_vm_readv(2)

```
commit 3af224c5cd8a64a6af3f875549ff821e2b5cb211
Author: Denys Vlasenko <vda.linux@googlemail.com>
Date:   Sat Jan 28 01:46:33 2012 +0100

    Use process_vm_readv instead of PTRACE_PEEKDATA to read data blocks

    Currently, we use PTRACE_PEEKDATA to read things like filenames and
    data passed by I/O syscalls.
    PTRACE_PEEKDATA gets one word per syscall. This is VERY expensive.
    For example, in order to print fstat syscall, we need to perform
    more than twenty trips into kernel to fetch one struct stat!

    Kernel 3.2 got a new syscall, process_vm_readv(), which can be used
    to copy data blocks out of process' address space.

    This change uses it in umoven() and umovestr() functions if possible,
    with fallback to old method if process_vm_readv() fails.
    If it returns ENOSYS, we don't try to use it anymore, eliminating
    overhead of trying it on older kernels.

    Result of "time strace -oLOG ls -l /usr/lib >/dev/null":
    before patch: 0.372s
    After patch:  0.262s
```

Syscall data retrieval

Cache retrieved data

```
commit e99ac2bd2b055c7804d22e3519d7ba23c8f34df8
Author: Dmitry V. Levin <ldv@altlinux.org>
Date:   Sun Sep 15 15:47:01 2019 +0000

    Implement memory caching for umove* functions

    When the data to be fetched by vm_read_mem resides in a single memory
    page, fetch the whole page and cache it.  This implementation caches
    up to two memory pages.
```

General optimisations

Tracee descriptor search

```
commit e8cb814cf23dad36319205447eddb857a98889a2
Author: Dmitry V. Levin <ldv@altlinux.org>
Date:   Tue Mar 1 14:42:58 2016 +0000

    Optimize pid2tcb

    Introduce an internal cache of pid2tcb translations.
    This can save more than 80% of CPU user time spent by strace.

        [...]

        old$ ./set_ptracer_any ./pid2tcb >pid2tcb.wait & \
          while [ ! -s pid2tcb.wait ]; do sleep 0.1; done; \
          time -f '%Uuser %Ssystem %eelapsed %PCPU' \
          ../strace -qq -enone -esignal=none -f -p $!
        5.51user 104.90system 122.45elapsed 90%CPU

        new$ ./set_ptracer_any ./pid2tcb >pid2tcb.wait & \
          while [ ! -s pid2tcb.wait ]; do sleep 0.1; done; \
          time -f '%Uuser %Ssystem %eelapsed %PCPU' \
          ../strace -qq -enone -esignal=none -f -p $!
        1.29user 102.78system 114.97elapsed 90%CPU
```

Tracee processing fairness fix

The initial upstreaming attempt

- In the end of the year 2008, a bug titled "Some threads stop when strace with -f option is executed on a multi-thread process" was reported via Red Hat Bugzilla

[1]https://gitlab.com/strace/strace/commit/215cc270
[2]https://gitlab.com/strace/strace/commit/f9a7e63a
[3]https://gitlab.com/strace/strace/commit/2c8a2583
[4]https://gitlab.com/strace/strace/commit/47ce6dfc
[5]https://lists.strace.io/pipermail/strace-devel/2009-February/000909.html
[6]https://lists.strace.io/pipermail/strace-devel/2009-May/001038.html
[7]https://lists.strace.io/pipermail/strace-devel/2009-June/001054.html
[8]https://gitlab.com/strace/strace/commit/eb9e2e89

Tracee processing fairness fix: reproducer

```
static int thd_no;

static void *sub_thd(void *c)
{
        for (;;)
                getuid();
        return NULL;
}

int main(int argc, char **argv)
{
        int i;
        pthread_t *thd;
        int num_threads = 10;

        thd = malloc(num_threads * sizeof(thd[0]));
        for (i = 0; i < num_threads; i++)
                pthread_create(&thd[i], NULL, sub_thd, NULL);

        return 0;
}
```

Tracee processing fairness fix: performance

`tests/looping_threads.test` (rewritten `test/many_looping_threads`), before the patch

```
2 threads,   0.00 user,  0.00 system,   0:00.00 elapsed, 80% CPU
3 threads,   0.31 user,  2.36 system,   0:01.58 elapsed, 169% CPU
4 threads,   0.08 user,  0.49 system,   0:00.38 elapsed, 152% CPU
5 threads,   4.74 user,  40.52 system,  0:25.23 elapsed, 179% CPU
6 threads,   19.27 user, 156.55 system, 1:41.25 elapsed, 173% CPU
```

Tracee processing fairness fix

The initial upstreaming attempt

- In the end of the year 2008, a bug titled "Some threads stop when strace with -f option is executed on a multi-thread process" was reported via Red Hat Bugzilla
- A series of patches was committed to the strace CVS repository[1][2][3][4] in the beginning of 2009 as a fix
- However, due to apparent disagreements with the maintainer[5][6][7], the patches have been reverted later that year[8]
- As a result, the fix has become RHEL-only

[1] https://gitlab.com/strace/strace/commit/215cc270
[2] https://gitlab.com/strace/strace/commit/f9a7e63a
[3] https://gitlab.com/strace/strace/commit/2c8a2583
[4] https://gitlab.com/strace/strace/commit/47ce6dfc
[5] https://lists.strace.io/pipermail/strace-devel/2009-February/000909.html
[6] https://lists.strace.io/pipermail/strace-devel/2009-May/001038.html
[7] https://lists.strace.io/pipermail/strace-devel/2009-June/001054.html
[8] https://gitlab.com/strace/strace/commit/eb9e2e89

Tracee processing fairness fix [2/4]

The following tumbling

- The issue has been discussed again in 2012[1], but no decision has been made
- Meanwhile, the fix has been forward-ported to RHEL 6 in 2009 (on top of 4.5.19) and then RHEL 7 (on top of 4.8 in 2012, on top of 4.10 in 2015, on top of 4.12 in 2016, and on top of 4.17 in 2017)
- When the need has arisen to forward-port the patch for strace 4.24 in 2018, yet another attempt to upstream the patch has been made
- The effort took more than half a year, but the patch has finally been included in strace 5.0, released on March 19th, 2019.

[1] `https://lists.strace.io/pipermail/strace-devel/2012-May/002306.html`

Tracee processing fairness fix

The final upstreamed commit

```
commit e0f0071b36215de8a592bf41ec007a794b550d45
Author:     Eugene Syromyatnikov <evgsyr@gmail.com>
AuthorDate: Wed Aug 8 21:41:39 2018 +0200
Commit:     Dmitry V. Levin <ldv@altlinux.org>
CommitDate: Wed Mar 6 23:20:39 2019 +0000

    Implement queueing of threads before dispatching them

    [...]

    Resolves: https://bugzilla.redhat.com/show_bug.cgi?id=478419
    Resolves: https://bugzilla.redhat.com/show_bug.cgi?id=526740
    Resolves: https://bugzilla.redhat.com/show_bug.cgi?id=851457
    Resolves: https://bugzilla.redhat.com/show_bug.cgi?id=1609318
    Resolves: https://bugzilla.redhat.com/show_bug.cgi?id=1610774
    Co-Authored-by: Dmitry V. Levin <ldv@altlinux.org>
    Co-Authored-by: Denys Vlasenko <dvlasenk@redhat.com>
    Co-Authored-by: Andreas Schwab <aschwab@redhat.com>
    Co-Authored-by: Jeff Law <law@redhat.com>
    Co-Authored-by: DJ Delorie <dj@redhat.com>

3 files changed, 283 insertions(+), 125 deletions(-)
```

Tracee processing fairness fix

The missing contributors

- `Jan Stancek <jstancek@redhat.com>`
- *. . . someone else?*

Tracee processing fairness fix: performance

tests/looping_threads.test (rewritten test/many_looping_threads), before the patch

```
2 threads,   0.00 user,  0.00 system,   0:00.00 elapsed, 80% CPU
3 threads,   0.31 user,  2.36 system,   0:01.58 elapsed, 169% CPU
4 threads,   0.08 user,  0.49 system,   0:00.38 elapsed, 152% CPU
5 threads,   4.74 user,  40.52 system,  0:25.23 elapsed, 179% CPU
6 threads,   19.27 user, 156.55 system, 1:41.25 elapsed, 173% CPU
```

tests/looping_threads.test, after the patch

```
2 threads,   0.00 user,  0.00 system,   0:00.00 elapsed, 80% CPU
34 threads,  0.00 user,  0.03 system,   0:00.02 elapsed, 173% CPU
66 threads,  0.01 user,  0.11 system,   0:00.06 elapsed, 180% CPU
[...]
418 threads, 0.43 user,  7.64 system,   0:06.11 elapsed, 132% CPU
450 threads, 0.47 user,  8.64 system,   0:06.83 elapsed, 133% CPU
482 threads, 0.55 user,  10.15 system,  0:08.04 elapsed, 133% CPU
514 threads, 0.63 user,  12.08 system,  0:09.63 elapsed, 132% CPU
546 threads, 0.72 user,  14.33 system,  0:11.57 elapsed, 130% CPU
578 threads, 0.80 user,  16.44 system,  0:13.25 elapsed, 130% CPU
610 threads, 0.92 user,  20.09 system,  0:16.51 elapsed, 127% CPU
```

IBM POWER 9 02CY089, 8 cores, 32 threads

dd slowdown

dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 0.128348 s, 4.0 MB/s
```

strace -eaccept dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 17.5274 s, 29.2 kB/s
```

Intel Sandy Bridge-EP (i7-3960X)
$\approx$ 136.5x slowdown

Some less artificial example[1]

Re-building of strace itself

Non-traced run	39.849 s
Under strace 4.6	121.840 s (3.06x slowdown)
Under strace 5.4	110.255 s (2.77x slowdown)

[1] `https://fosdem.org/2018/schedule/event/debugging_tools_stracing_build/`

dd slowdown

dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 0.128348 s, 4.0 MB/s
```

strace -eaccept dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 17.5274 s, 29.2 kB/s
```

Intel Sandy Bridge-EP (i7-3960X)
$\approx 136.5x$ slowdown

Microarchitectural side-channel attack mitigations

dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 0.540779 s, 947 kB/s
```

strace -eaccept dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 20.2093 s, 25.3 kB/s
```

Intel Sandy Bridge-EP (i7-3960X) *with patched kernel and microcode (up to MDS)*
$\approx 37.37x$ slowdown

dd slowdown: mips

dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 1.75059 s, 292 kB/s
```

strace -eaccept dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 66.6589 s, 7.7 kB/s
```

Cavium Octeon CN5020
$\approx 38.07x$ slowdown

dd slowdown: armv7

dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 0.732232 s, 699 kB/s
```

strace -eaccept dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 42.3294 s, 12.1 kB/s
```

Broadcom BCM2835
$\approx 57.8x$ slowdown

dd slowdown: ppc64le

dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 0.480209 s, 1.1 MB/s
```

strace -eaccept dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 18.0921 s, 28.3 kB/s
```

IBM POWER 9 02CY089
$\approx 37.67x$ slowdown

dd slowdown: s390x

dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 0.221291 s, 2.3 MB/s
```

strace -eaccept dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 6.58973 s, 77.7 kB/s
```

IBM System z12, 2-core LPAR
$\approx 29.77x$ slowdown

Microarchitectural side-channel attack mitigations

dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 0.540779 s, 947 kB/s
```

strace -eaccept dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 20.2093 s, 25.3 kB/s
```

Intel Sandy Bridge-EP (i7-3960X) *with patched kernel and microcode (up to MDS)*
$\approx 37.37x$ slowdown

Microarchitectural side-channel attack mitigations

dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 0.540779 s, 947 kB/s
```

strace -eaccept dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 20.2093 s, 25.3 kB/s
```

Intel Sandy Bridge-EP (i7-3960X) *with patched kernel and microcode (up to MDS)*
$\approx 37.37x$ slowdown

But we still have to stop a tracee on each syscall.

seccomp-assisted system call filtering [1/2]

Ah, no, we haven't.

seccomp-bpf usage

- `seccomp` (for Secure Computing) is a Linux mechanism that provides an ability (in its `SECCOMP_SET_MODE_FILTER` mode) to attach a BPF program to a process
- Since Linux 3.5, a `seccomp` program has an ability to return `SECCOMP_RET_TRACE` as a result of its execution, which, in turn, notifies `ptrace`-based tracer with `PTRACE_EVENT_SECCOMP`
- When `--seccomp-bpf` command-line option is passed to `strace`, BPF bytecode is generated and attached to tracees, if possible
- The feature has been implemented as part of GSoC 2018 and 2019 projects by Chen Jingpiao and Paul Chaignon, and included in the 5.3 release of `strace`

seccomp-assisted system call filtering

dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000 bytes (512 kB) copied, 0.540779 s, 947 kB/s
```

strace -eaccept dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000 bytes (512 kB) copied, 20.2093 s, 25.3 kB/s
```

strace **--seccomp-bpf** -eaccept dd if=/dev/zero of=/dev/null bs=1 count=500k

```
512000 bytes (512 kB) copied, 0.79769 s, 642 kB/s
```

$\approx 37.37x$ slowdown before, $\approx 1.475x$ after, $\approx 25.33x$ improvement

Some less artificial example[1]

Re-building of strace itself

Non-traced run	: 39.849 s
Under strace 4.6	: 121.840 s (3.06x slowdown)
Under strace 5.4	: 110.255 s (2.77x slowdown)
Under strace 5.4 with seccomp-bpf	: 57.520 s (1.44x slowdown)

[1] `https://fosdem.org/2018/schedule/event/debugging_tools_stracing_build/`

Future plans

Near future

- Enable seccomp-bpf by default
- Refine tracee's memory caching strategy

Distant future

- `strace`'s seccomp-bpf filter would benefit greatly from eBPF maps, once they are made available for seccomp eBPF programs
- Moving away from `ptrace` tracing backend is a possibility, once some other tracing backend (perf?) would allow stopping tracee instead of dropping events[1][2]

[1] See `https://linuxplumbersconf.org/event/2/contributions/78/`, "Problem 8: strace is slow, perf can lose data"

[2] `https://lore.kernel.org/lkml/20181128134700.212ed035@gandalf.local.home/`

Questions?

homepage

https://strace.io/

strace.git

https://gitlab.com/strace/strace.git
https://github.com/strace/strace.git

mailing list

strace-devel@lists.strace.io

IRC channel

#strace@freenode