CAN STRACE MAKE YOU FAIL?

Nahim El Atmani | @brokenpi_pe

July 15, 2016

DEFINITION 1.0

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

TESTS

codebases sometime have:

- complex code paths (because of the size, design choices...)
- untested error handlers

FAULT INJECTION TO THE RESCUE

In software testing, fault injection is a technique for improving the coverage of a test by introducing faults to test code paths, in particular error handling code paths, that might otherwise rarely be followed.

BENEFITS

- Better tests
- Better coverage
- Fuzzing?

STRACE FAULT INJECTION

Here we tamper with syscalls, thanks to *ptrace(2)*.
Inspiration comes from seccomp:

```
/* arch/x86/entry/common.c */
unsigned long
syscall_trace_enter_phase1(struct pt_regs *regs, u32 arch)
{
    // [...]
    if (ret == SECCOMP_PHASE1_SKIP) {
        regs->orig_ax = -1;
        ret = 0;
    } else if (ret != SECCOMP_PHASE1_OK) {
        return ret;  /* Go directly to phase 2 */
    // [...]
}
```

Changing syscall number on the fly by -1 vaporizes it... 👍

STRACE INTERNALS (1/2)

- Using *PTRACE_SYSCALL* strace get {pre,post}-syscall hook
- Around this simple concept tons of bookkeeping information are kept

```
static int
trace_syscall_entering(struct tcb *tcp)
{
    /*
     * Conditionaly setup the faulting state
     * Call the arch dependant part to discard the syscall
     */
```

```
static int
trace_syscall_exiting(struct tcb *tcp)
{
    /*
     * Clear the faulting state
     * Call the arch dependant part to set the correct errno
```

ARCH DEPENDANT

The trick does require some arch dependant code to modify registers accordingly: Here is the x86_64 version of it:

```
long
fault_discard_sc(struct tcb *tcp) {
    return ptrace(PTRACE_POKEUSER, tcp->pid,
                  offsetof(struct user, regs.orig_rax),
                  (unsigned long)-1);
}
```

POC

Sample program, no tampering:

```
build/strace -e kill ./dummy
Can you get the flag before I kill myself?
kill(3802, SIGKILL <unfinished ...>
+++ killed by SIGKILL +++
```

Preventing suicide?

```
Can you get the flag before I kill myself?
kill(4320, SIGKILL) = -1 EINVAL (DISCARDED)
lse_week{Faults_Injection_Can_Save_lives}
+++ exited with 0 +++
```

DEFINITION 1.1

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.*

FILTERS (1/2)

Let's say we **don't want** our target to be able to **open** the first file it opens. We **can't** discard every *open(2)* out there...

hello.c (ie without *open(2)* in the code)

```
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = -1
open("/usr/lib/tls/x86_64/libpthread.so.0", O_RDONLY|O_CLOEXEC) = -1
open("/usr/lib/tls/libpthread.so.0", O_RDONLY|O_CLOEXEC) = -1
open("/usr/lib/x86_64/libpthread.so.0", O_RDONLY|O_CLOEXEC) = -1
open("/usr/lib/libpthread.so.0", O_RDONLY|O_CLOEXEC) = -1
./hello: error while loading shared libraries: libpthread.so.0:
cannot open shared object file: Invalid argument
+++ exited with 127 +++
```

FILTERS (2/2)

✔	✖
nth	after nth
every nth	before nth
n%	…

Mainly parsing & tests…

BUGS FOUND?

Yes. The first real world test I made was with Python 3.5.1:

```
Fatal Python error: Failed to open /dev/urandom
--- SIGSEGV {si_signo=SIGSEGV, si_code=SEGV_MAPERR, si_addr=0x50} ---
+++ killed by SIGSEGV (core dumped) +++
```

```
Message: Process 9427 (python) of user 1000 dumped core.

         Stack trace of thread 9427:
         #0  0x00000000005015a3 PyErr_Fetch (python)
         #1  0x000000000051fdc6 _Py_PrintFatalError (python)
         #2  0x0000000000520099 Py_FatalError (python)
         #3  0x0000000000520d6a dev_urandom_noraise (python)
         #4  0x000000000052122f _PyRandom_Init (python)
         #5  0x000000000041e119 Py_Main (python)
         #6  0x000000000041a71f main (python)
         #7  0x00007fb51daa5741 __libc_start_main (libc.so.6)
         #8  0x000000000041a449 _start (python)
```

Does Py_FatalError() throw a SIGSEGV? Intuitively, no.

PYTHON3 (1/3)

Seems something went wrong in the middle of it because:

```
/* Print fatal error message and abort */

void
Py_FatalError(const char *msg)
{
    /* [...] */
    exit:
    #if defined(MS_WINDOWS) && defined(_DEBUG)
        DebugBreak();
    #endif
        abort();
}
```

PYTHON3 (2/3)

Found NULL dereferences because no checks are made on the return of *PyThreadState_GET()*

```
void
PyErr_Fetch(PyObject **p_type, PyObject **p_value, PyObject **p_traceback
{
    PyThreadState *tstate = PyThreadState_GET();

    *p_type = tstate->curexc_type; /* <-- HERE */
    *p_value = tstate->curexc_value; /* <-- AND HERE */
    *p_traceback = tstate->curexc_traceback; /* <-- OR HERE */

    tstate->curexc_type = NULL; /* <-- ALSO HERE */
    tstate->curexc_value = NULL; /* <-- STILL HERE */
    tstate->curexc_traceback = NULL; /* <-- You get it... */
}
```

PYTHON3 (3/3)

corrected between Python 3.5.1 & Python 3.6.0a2+

```
getrandom(0x916930, 24, GRND_NONBLOCK)  = -1 EINVAL (DISCARDED)
Fatal Python error: getrandom() failed

--- SIGABRT {si_signo=SIGABRT, si_code=SI_TKILL, si_pid=4127, si_uid=10(
+++ killed by SIGABRT (core dumped) +++
```

They also discovered *getrandom()* from  v3.17-rc1 and reading raw */dev/urandom* is now a fallback.

TRY IT YOURSELF!

Because it's fun, sometimes it's also worth some $$:

- Search for common mistakes
- wait for the fuzzy logic to be implemented

So what is a common mistake?

BLIND TRUST (1/2)

> If *open(2)* worked, *fstat(2)* will.

Nope.

```
struct stat fs;

int main(int arc, char *argv[])
{
    /* ... */
    if ((fd = open(argv[1], O_RDONLY )) < 0)
        perror("main");

    fstat(fd, &fs);
    printf("%ld\n", file_size);
    do_file_histogram(fd, hist);
    for (int i = 0; i < sizeof(hist); ++i)
        hist[i] /= fs.st_size; /* Histogram normalization */
    /* ... */

    return 0;
}
```

BLIND TRUST (2/2)

Here is the result of abusive trust:

```
$ strace -a0 -e fstat -e faultwith=fstat:3:EINVAL ./fstat tags
fstat(4, {st_mode=S_IFREG|0644, st_size=257742, ...}) = 0
fstat(4, {st_mode=S_IFREG|0755, st_size=1960968, ...}) = 0
fstat(4, 0x600c00) = -1 EINVAL (DISCARDED)
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 11), ...}) = 0
--- SIGFPE {si_signo=SIGFPE, si_code=FPE_INTDIV, si_addr=0x400734} ---
+++ killed by SIGFPE (core dumped) +++
[2]    8604 floating point exception (core dumped)
```

Remember?

```
for (int i = 0; i < sizeof(hist); ++i)
    hist[i] /= fs.st_size; /* Divide by zero (fs was in the bss) */
```

META IMPR:::VMENT

- In strace the main work is parsing
- So you have to test your parser
- How do you test tricky syscalls like *reboot(2)*?
- How to test an *ioctl* without having the behavior?

ENOMANA

```
int reboot(int magic, int magic2, int cmd, void *arg);
```

```
ERRORS
       [...]

       EINVAL Bad magic numbers or cmd.
```

- It does not seem very efficient to actually issue a *reboot(2)* to test it.
- With the fault injection we can actually parse and test it, hence improving the code coverage

▢ IOCTL BLACK BOX

- We can't use -1 as a file descriptor
- So we need a real file descriptor
- But we don't want the real behaviors
- What can we do?

CONCLUSION

- A lot of flaky, often untested patterns in the wild
- Fun and promising strace option
- some work is still needed for simpler usability

QUESTION

@brokenpi_pe

https://brokenpi.pe

nahim+dev@naam.me

>_ Naam@irc.rezosup.org

30 sec