Fault Injection in System Calls

Angelo Haller

2015-05-28

Why System Calls?

- Interface between applications and the kernel
- 'Dependability' of the kernel:
  denial of service, privilege escalation, ...
- Large surface area, reaching deep into underlying kernel structures
  $>$ 300 system calls in Linux 3.10
- New features added on a regular basis

`open, read, write, close, mmap, ioctl, mlock, ...`

Trinity

Semi-intelligent Linux system call fuzzer

- Developed primarily by Dave Jones since 2006; gained momentum in 2010
- Free and open source (GPLv2)
- Actively used to fuzz current kernel releases
- Found more than 150 bugs in 2012 alone

Trinity - Bugs Found

- Oldest dates back to 1996: kernel oops when calling setsockopt
- CVE-2010-4256: pipe_fcntl local DoS
- CVE-2010-4169: use-after-free vulnerability in mm/mprotect.c
- CVE-2011-1748: raw_release in net/can/raw.c NULL pointer dereference
- world writable acpi file in sysfs
- ...

Trinity - Inner Workings

Knowledge about system call arguments encoded in *annotation files*:

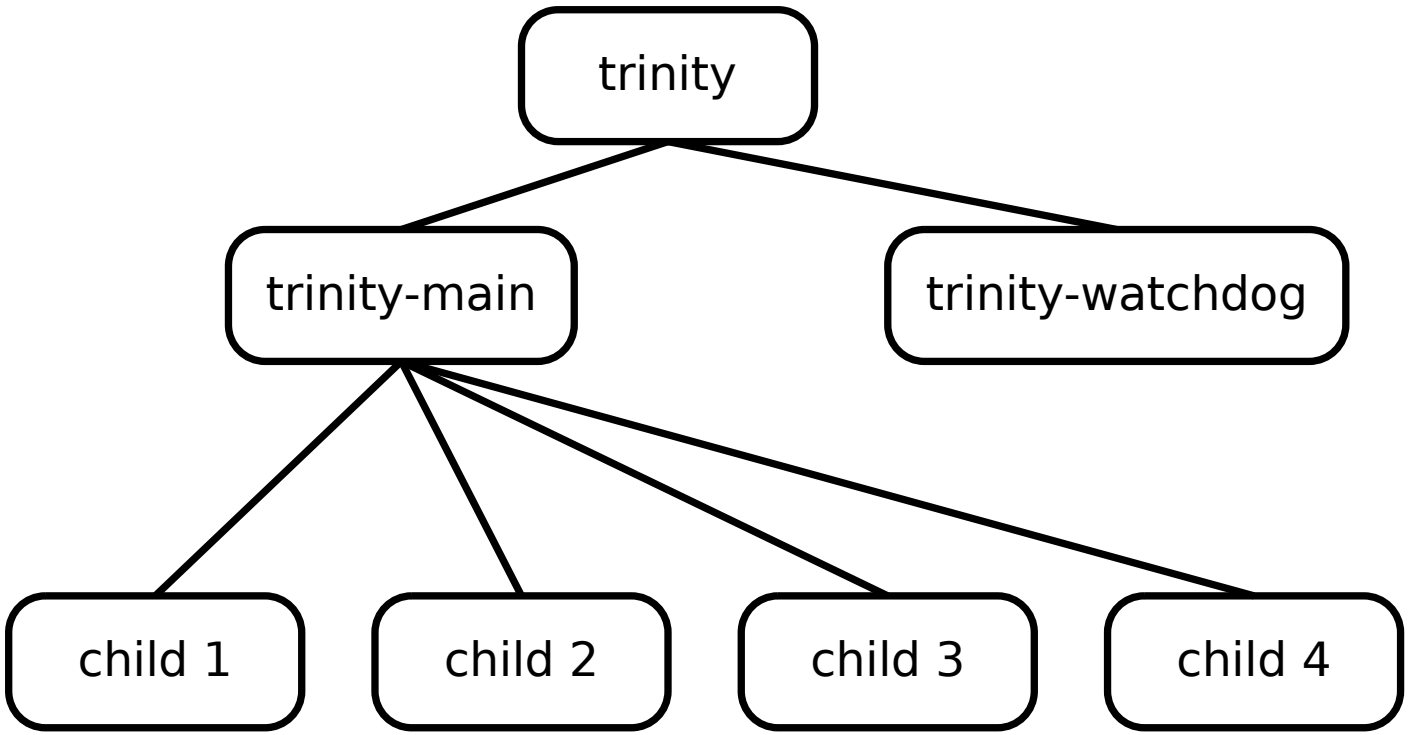
- buffers & buffer sizes
- file descriptors
- addresses
- process IDs
- custom methods to sanitise arguments

Valid values (e.g. file descriptors) are created on startup and then shared among fuzzing processes.

Trinity - Inner Workings

- Annotations regarded most of the time to pass early checks within system calls
- Random bit flips disregarding annotations to probe system call sanity checks
- Fuzzing of 'interesting' values:
  off by one, misaligned addresses, kernel/user space addresses, ...
- Limited support for seeds

Trinity - Fuzzing Process



Annotated System Call: open

```
static void sanitise_open(struct syscallrecord *rec);

struct syscallentry syscall_open = {
  .name = "open",
  .num_args = 3,
  .arg1name = "filename",
  .arg1type = ARG_PATHNAME,
  .arg2name = "flags",
  .arg2type = ARG_OP,
  .arg2list = {
    .num = 4,
    .values = { O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, },
  },
  .arg3name = "mode",
  .arg3type = ARG_MODE_T,
  .sanitise = sanitise_open,
};
```

Demo

```
[main] fd[662] = fopen /proc/651/net/sctp (read-only) flags:0 fcntl_flags:0
[main] fd[663] = fopen /proc/713/smaps (read-only) flags:0 fcntl_flags:400
[main] fd[664] = fopen /sys/devices/virtual/net/nr2/statistics/tx_aborted_errors
 (read-only) flags:0 fcntl_flags:4000
[main] fd[665] = timerfd
[main] fd[666] = timerfd
[main] fd[667] = timerfd
[main] fd[668] = timerfd
[main] fd[669] = timerfd
[main] fd[670] = timerfd
[main] fd[671] = timerfd
[main] fd[672] = timerfd
[main] fd[673] = open("trinity-testfile1", flags:105000)
[main] fd[674] = fopen("trinity-testfile2", O_RDWR)
[main] fd[675] = open("trinity-testfile3", flags:105000)
[main] fd[676] = fopen("trinity-testfile4", O_RDWR)
[main] Enabled 9 fd providers.
[watchdog] Watchdog is alive. (pid:714)
[watchdog] 48604 iterations. [F:31136 S:17467 HI:14651]
[watchdog] kernel became tainted! (128/0) Last seed was 256421206
[main] Bailing main loop because kernel became tainted..
[watchdog] [714] Watchdog exiting because kernel became tainted..
[init] Ran 48604 syscalls. Successes: 17468  Failures: 31136
```

```
./trinity -q
[init] shm is at 0x7f7efa034000
[init] Using user passed random seed: 256421206
[init] 32-bit syscalls: 359 enabled, 0 disabled.  64-bit syscalls: 323
[init] mapping[0]: (zeropage PROT_READ | PROT_WRITE) 0x7f7efa006000 (8
[...]
[main] fd[121] = domain:4 (PF_IPX) type:0x2 protocol:48
[main] setsockopt(1 a 1f8dc10 c7) on fd 121 [4:2:48]
[main] fd[122] = domain:5 (PF_APPLETALK) type:0x2 protocol:0
[...]
[main] Generating file descriptors
[main] Added 159 filenames from /dev
[main] Added 12154 filenames from /proc
[main] Added 8981 filenames from /sys
[main] fd[416] = open /sys/devices/pci0000:00/0000:00:03.0/device (rea
[main] fd[417] = fopen /proc/41/net/irda/discovery (read-only) flags:0
[...]
[main] Enabled 9 fd providers.
[watchdog] Watchdog is alive. (pid:982)
[watchdog] kernel became tainted! (512/0) Last seed was 256421206
[main] Bailing main loop because kernel became tainted..
[watchdog] [982] Watchdog exiting because kernel became tainted..
[init] Ran 1224 syscalls. Successes: 239  Failures: 985
```

```
less tmp/trinity-post-mortem.log
[child0:998] [127] getresuid(ruid=0x7f7ef9307000,
    euid=0x7f7ef9307001,
    suid=0xffffffffffbffff) = -1 (Bad address)
[child0:998] [128] clock_nanosleep(which_clock=0x2,
    flags=0x0, rqtp=0x7f7ef9307000,
    rmtp=0xffffffffff004000)
```

Fault Injection - Model [Cristian]

$\Rightarrow$ Mainly aimed at **crash faults**

Trigger and debug of other faults possible too:

- Omission faults
- Computation faults
- Timing faults

Fault Injection - Implementation

- Trigger mechanism
  Location based
- Injection time
  During runtime
- Injection level
  Assembler

Faults are injected into the CPU registers before performing the system call

Shortcomings

- Manual annotation of system calls
- Leaks memory
- Can not fuzz all system calls without interfering with the tool itself (e.g. `munmap`, `shutdown`)
- System calls not ordered (interdependencies)
- Reproducibility: race conditions, dependencies
- Difficult to find actual fault location in source

Similar Tools

iknowthis (Google)

- Very similar to Trinity: annotated files
- Supports Linux and FreeBSD
- Seems to be abandoned since 2012

sysfuzz

- Very similar to Trinity: annotated files
- FreeBSD only

Discussion

References I

S. Winter, C. Sârbu, N. Suri, and B. Murphy, "The impact of fault models on software robustness evaluations," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11, Waikiki, Honolulu, HI, USA: ACM, 2011, pp. 51–60, ISBN: 978-1-4503-0445-0. DOI: 10.1145/1985793.1985801. [Online]. Available: http://doi.acm.org/10.1145/1985793.1985801.

M. Kerrisk. (Feb. 6, 2013), Lca: The trinity fuzz tester, [Online]. Available: https://lwn.net/Articles/536173/ (visited on May 22, 2015).

D. Jones, *Trinity: A linux kernel fuzz tester.* Linux Conf Australia, 2013. [Online]. Available: http://codemonkey.org.uk/projects/talks/LCA2013-Trinity.pdf.

Syscalls (2), Apr. 17, 2013. [Online]. Available: https://www.freebsd.org/cgi/man.cgi?query=syscalls&sektion=2&manpath=CentOS+7.1 (visited on May 22, 2015).

——, (Nov. 9, 2010), System call abuse, [Online]. Available: http://codemonkey.org.uk/2010/11/09/system-call-abuse/ (visited on May 22, 2015).

——, (Dec. 15, 2010), System call fuzzing continued, [Online]. Available: http://codemonkey.org.uk/2010/12/15/system-call-fuzzing-continued/ (visited on May 22, 2015).

References II

——, *Trinity 1.5*, Mar. 2, 2015. [Online]. Available: http://codemonkey.org.uk/projects/trinity/trinity-1.5.tar.xz (visited on May 2, 2015).

T. Ormandy. (Sep. 14, 2011), Iknowthis, [Online]. Available: https://code.google.com/p/iknowthis/ (visited on May 27, 2015).

M. Johnston. (Nov. 16, 2014), Sysfuzz, [Online]. Available: https://github.com/markjdb/sysfuzz (visited on May 27, 2015).

C. Evans and T. Ormandy. (Aug. 25, 2014), The poisoned nul byte, 2014 edition, [Online]. Available: http://googleprojectzero.blogspot.de/2014/08/the-poisoned-nul-byte-2014-edition.html (visited on May 27, 2015).