Trinity

A special fuzzer aimed merely at Linux Kernel

Yong Sun
QA APACII
yosun@suse.com

Agenda

- Introduce Trinity
- Syscalls
- Demo
- Real bugs find by Trinity

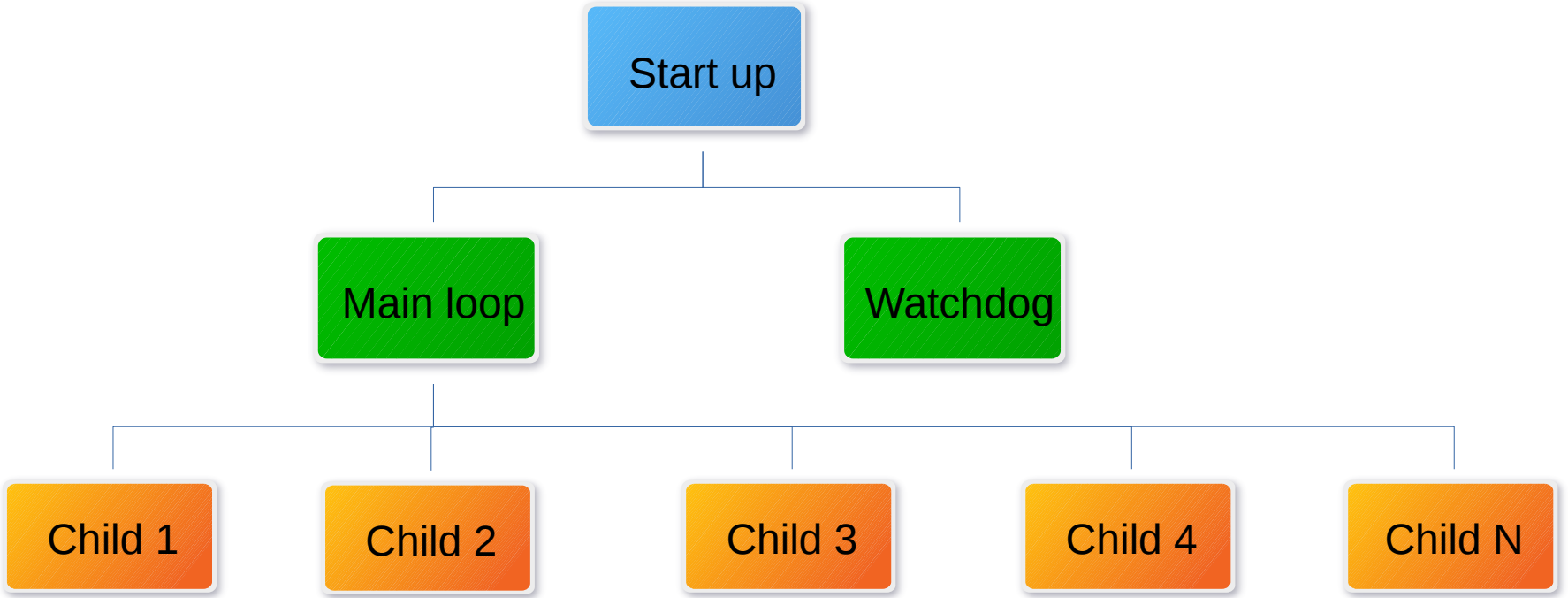
What is Trinity

Introduce Trinity

Goals

- Focus on testing Linux Kernel
- Use all the fuzzing strategies
- Be format agnostic
- Be fast

Trinity Process model



Watchdog

- Keeps track of child progress.
- SIGKILL if 'stuck'.
- Sanity check shared memory.

Syscall Trinity supported

Process Control

- fork 创建一个新进程
- clone 按指定条件创建子进程
- execve 运行可执行文件
- exit 中止进程
- getpgid 获取指定进程组标识号
- getpid 获取进程标识号
- getppid 获取父进程标识号
- getpriority 获取调度优先级
- setpriority 设置调度优先级
- nanosleep 使进程睡眠指定的时间
- nice 改变分时进程的优先级
- pause 挂起进程，等待信号
- personality 设置进程运行域
- prctl 对进程进行特定操作
- ptrace 进程跟踪
- sched_get_priority_max 取得静态优先级的上限
- sched_get_priority_min 取得静态优先级的下限
- sched_getparam 取得进程的调度参数
- sched_getscheduler 取得指定进程的调度策略
- sched_rr_get_interval 取得按RR算法调度的实时进程的时间片长度
- sched_setparam 设置进程的调度参数
- sched_setscheduler 设置指定进程的调度策略和参数
- sched_yield 进程主动让出处理器，并将自己等候调度队列队尾
- vfork 创建一个子进程，以供执行新程序，常与execve等同时使用
- wait 等待子进程终止
- waitpid 等待指定子进程终止
- wait4 参见waitpid
- capget 获取进程权限
- capset 设置进程权限
- getsid 获取会晤标识号
- setsid 设置会晤标识号

File System Control

[File read and write behavior]
fcntl 文件控制
open 打开文件
creat 创建新文件
close 关闭文件描述字
read 读文件
write 写文件
lseek 移动文件指针
llseek 在64位地址空间里移动文件指针
dup 复制已打开的文件描述字
flock 文件加/解锁
poll I/O多路转换
truncate 截断文件
ftruncate 参见truncate
umask 设置文件权限掩码
fsync 把文件在内存中的部分写回磁盘

[File system behavior]
access 确定文件的可存取性
chdir 改变当前工作目录
fchdir 参见chdir
chmod 改变文件方式
fchmod 参见chmod
chown 改变文件的属主或用户组
fchown 参见chown
lchown 参见chown
chroot 改变根目录
stat 取文件状态信息
lstat 参见stat
fstat 参见stat
statfs 取文件系统信息
fstatfs 参见statfs

readdir 读取目录项
getdents 读取目录项
mkdir 创建目录
mknod 创建索引节点
rmdir 删除目录
rename 文件改名
link 创建链接
symlink 创建符号链接
unlink删除链接
readlink 读符号链接的值
mount 安装文件系统
umount 卸下文件系统
ustat 取文件系统信息
utime 改变文件的访问修改时间
utimes 参见utime
quotactl 控制磁盘配额

System Control

ioctl　I/O 总控制函数
sysctl 读 / 写系统参数
acct　启用或禁止进程记账
getrlimit　获取系统资源上限
setrlimit　设置系统资源上限
getrusage　获取系统资源使用情况
uselib 选择要使用的二进制函数库
reboot　重新启动
bdflush　控制 bdflush 守护进程
sysfs 取核心支持的文件系统类型
sysinfo　取得系统信息

adjtimex 调整系统时钟
alarm 设置进程的闹钟
getitimer 获取计时器值
setitimer 设置计时器值
gettimeofday　取时间和时区
settimeofday　设置时间和时区
stime 设置系统日期和时间
time　取得系统时间
times 取进程运行时间
uname　获取当前 UNIX 系统的名称、版本和主机等信息
vhangup 挂起当前终端
nfsservctl　对 NFS 守护进程进行控制
delete_module 删除可装载的模块项

Memory Control & Network Control

brk　改变数据段空间的分配
mlock内存页面加锁
munlock 内存页面解锁
mlockall 调用进程所有内存页面加锁
munlockall 调用进程所有内存页面解锁
mmap映射虚拟内存页
munmap 去除内存页映射
mremap 重新映射虚拟内存地址
msync　将映射内存中的数据写回磁盘
mprotect 设置内存映像保护
getpagesize 获取页面大小
sync 将内存缓冲区数据写回硬盘

setdomainname　设置域名
sethostname　设置主机名称

Socket Control

socketcall　socket 系统调用
socket　建立 socket
bind　绑定 socket 到端口
connect　连接远程主机
accept　响应 socket 连接请求
send　通过 socket 发送信息
recv　通过 socket 接收信息
listen 监听 socket 端口

select 对多路同步 I/O 进行轮询
shutdown　关闭 socket 上的连接
getsockname　取得本地 socket 名字
getpeername　获取通信对方的 socket 名字
getsockopt　取端口设置
setsockopt　设置端口参数
sendfile　在文件或端口间传输数据
socketpair　创建一对已联接的无名 socket

User Control

getuid　获取用户标识号
setuid设置用户标志号
getgid　获取组标识号
setgid设置组标志号
getegid　获取有效组标识号
geteuid　获取有效用户标识号
getgroups　获取后补组标志清单
setgroups　设置后补组标志清单

setregid 分别设置真实和有效的的组标识号
setreuid 分别设置真实和有效的用户标识号
getresgid　分别获取真实的,有效的和保存过的组标识号
setresgid分别设置真实的,有效的和保存过的组标识号
getresuid　分别获取真实的,有效的和保存过的用户标识号
setresuid分别设置真实的,有效的和保存过的用户标识号
setfsgid 设置文件系统检查时使用的组标识号
setfsuid 设置文件系统检查时使用的用户标识号

Inter-process Communication

ipc进程间通信总控制调用

[Signal]
sigaction设置对指定信号的处理方法
sigprocmask　根据参数对信号集中的信号执行阻塞/解除阻塞等操作
sigpending　为指定的被阻塞信号设置队列
sigsuspend 挂起进程等待特定信号
signal参见 signal
kill向进程或进程组发信号
ssetmask　ANSI C的信号处理函数

[Pipeline]
pipe　创建管道

[Message]
msgctl　消息控制操作
msgget　获取消息队列
msgsnd　发消息
msgrcv　取消息

[Semaphore]
semctl　信号量控制
semget　获取一组信号量
semop　信号量操作

[Memery sharing]
shmctl　控制共享内存
shmget　获取共享内存
shmat连接共享内存
shmdt　拆卸共享内存

Demo

How to get Trinity

To use it, just add factory repo and install it by zypper

- zypper ar http://download.suse.de/ibs/SUSE:/Factory:/Head/standard/ factory
- zypper in trinity

Code

- https://github.com/kernelslacker/trinity
- git clone https://github.com/kernelslacker/trinity.git

Trinity with specific syscall (e.g writev)

Make a test dir, and run in **non-root user**

- useradd testuser

Run in a separate folder, because it will create lots of logs to where you run it.

- mkdir testFolder
- chown testuser testFolder; cd testFolder
- su testuser

-c will make trinity run in specific syscall

- trinity -c writev

trinity is all

Also run as non-root user

If you only run a simple”trinity” command without any parameters it will run all syscall test

- trinity

Run it in multi-CPU

trinity -C <number_of_cpus>

e.g. trinity -C 4

It forks the process that number of times and parallelism is always good to trigger bugs

Bsc#968063
kmemleak: 2 new suspected memory leaks in copy_thread_tls

trinity/syscalls/fork.c

```
22 static void post_fork(struct syscallrecord *rec)
23 {
24      pid_t pid;
25
26      pid = rec->retval;
27      if (pid == 0) {
28              // child
29              sleep(1);
30              _exit(EXIT_SUCCESS);
31      } else {
32              __unused__ int ret;
33
34              while (pid_alive(pid) == TRUE) {
35                      int status;
36                      ret = waitpid(pid, &status, WUNTRACED | WCONTINUED | WNOHANG);
37              }
38      }
39 }
40
41 struct syscallentry syscall_fork = {
42      .name = "fork",
43      .num_args = 0,
44      .flags = AVOID_SYSCALL, // No args to fuzz, confuses fuzzer
45      .post = post_fork,
46 };
                                                    46,1          Bot
```

Bug 968063 - [trinity] kmemleak: 2 new suspected memory leaks in copy_thread_tls

```
unreferenced object 0xffff8805917c8000 (size 8192):
  comm "fork-leak", pid 2932, jiffies 4295354292 (age 1871.028s)
  hex dump (first 32 bytes):
    ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ................
    ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ................
  backtrace:
    [<ffffffff815a5e66>] create_object+0x376/0x870
    [<ffffffff815a5af0>] create_object+0x0/0x870
    [<ffffffff815a60ee>] create_object+0x5fe/0x870
    [<ffffffff8103ab43>] copy_thread_tls+0x6c3/0x9a0
    [<ffffffff8157100d>] __kmalloc_track_caller+0xdd/0x190
    [<ffffffff814cfbf5>] kmemdup+0x25/0x50
    [<ffffffff8103ab43>] copy_thread_tls+0x6c3/0x9a0
    [<ffffffff81150174>] copy_process+0x1a84/0x5790
    [<ffffffff8114e6f0>] copy_process+0x0/0x5790
    [<ffffffff811dc375>] wake_up_new_task+0x2d5/0x6f0
    [<ffffffff8115411d>] _do_fork+0x12d/0x820
    [<ffffffff81153ff0>] _do_fork+0x0/0x820
    [<ffffffff81117cbc>] __do_page_fault+0x38c/0x870
    [<ffffffff81118235>] trace_do_page_fault+0x65/0x1f0
    [<ffffffff8228b06e>] entry_SYSCALL_64_fastpath+0x12/0x71
    [<ffffffffffffffff>] 0xffffffffffffffff
```

My suspect is fail path in copy_process, nothing from bad_fork_cleanup_io (inclusive) and above frees the bitmap memory.

Bsc#1001322
writev syscall caused a call trace on btrfs

trinity/syscalls/write.c

```
/*
 * SYSCALL_DEFINE3(writev, unsigned long, fd, const struct iovec __user *, vec, unsigned long, vlen)
 */

struct syscallentry syscall_writev = {
        .name = "writev",
        .num_args = 3,
        .arg1name = "fd",
        .arg1type = ARG_FD,
        .arg2name = "vec",
        .arg2type = ARG_IOVEC,
        .arg3name = "vlen",
        .arg3type = ARG_IOVECLEN,
        .flags = NEED_ALARM,
};
```

Bug 1001322 - [trinity]Btrfs:writev syscall caused a call trace on btrfs

While trinity'ing,there is a call trace in dmesg,details as follows:

```
[620479.675628] ------------[ cut here ]------------
...
[620479.675696] CPU: 1 PID: 25050 Comm: trinity-c1 Tainted: G                X 4.4.16-56-default #1
[620479.675696] Hardware name: Intel Corporation Shark Bay Client platform/Flathead Creek Crb, BIOS HSWLPTU1.86C.0086.R00.1208052028 08/05/2012
[620479.675698]  0000000000000000 ffffffff8130d290 0000000000000000 ffffffffa055e6ae
[620479.675699]  ffffffff8107c121 ffff88009b998000 0000000000009000 ffff88009ba46600
[620479.675700]  0000000000000794 00000000ffffffffb ffffffffa04c7e30 ffff8801d34e0598
[620479.675701] Call Trace:
[620479.675709]  [<ffffffff81019a59>] dump_trace+0x59/0x310
[620479.675713]  [<ffffffff81019dfa>] show_stack_log_lvl+0xea/0x170
[620479.675714]  [<ffffffff8101ab81>] show_stack+0x21/0x40
[620479.675718]  [<ffffffff8130d290>] dump_stack+0x5c/0x7c
[620479.675722]  [<ffffffff8107c121>] warn_slowpath_common+0x81/0xb0
[620479.675732]  [<ffffffffa04c7e30>] btrfs_free_reserved_data_space_noquota+0xe0/0xf0 [btrfs]
[620479.675744]  [<ffffffffa04c7e57>] btrfs_free_reserved_data_space+0x17/0x30 [btrfs]
[620479.675757]  [<ffffffffa04f4c35>] __btrfs_buffered_write+0x3a5/0x580 [btrfs]
[620479.675769]  [<ffffffffa04f8327>] btrfs_file_write_iter+0x2f7/0x530 [btrfs]
[620479.675774]  [<ffffffff811fa8f7>] do_readv_writev+0x1c7/0x280
[620479.675776]  [<ffffffff811fb696>] SyS_writev+0x46/0xc0
[620479.675780]  [<ffffffff815def6e>] entry_SYSCALL_64_fastpath+0x12/0x6d
[620479.677026] DWARF2 unwinder stuck at entry_SYSCALL_64_fastpath+0x12/0x6d

[620479.677027] Leftover inexact backtrace:

[620479.677049] ---[ end trace 100c65073989f68a ]---
```

Reproduce this bug – loop running

This run pass will make sure the problem has been verified **99%**

- for i in $(seq 100);do trinity -c writev;done

But still can't say that it's 100% verified, because trinity is **Randomly**

Real challenge:

- Need deep kernel knowledge of where the problem is.
- Need a little bit C program to write a test script to touch the bad part.

Next step

- Add Trinity into openQA (Kernel Regression test)
  Test both Physical machine & VM Guest
- Better way to reproduce Trinity bug

Summary

- Easy to Install/Run.
- Analysis by checking dmesg, messages or serial console.
- Loop run a specific syscall to reproduce bug.
- Hard to verify.

Refer to

- https://github.com/kernelslacker/trinity
- SLE12-SP2 Kernel Testing, Jiří Slabý, SUSE Lab Conf 2016
  http://events.suse.cz/labs2016/slides/jslaby.pdf