Fault Tolerant System Calls

Luca Kleinschmidt
Supervised by Lukas Pirl

17.02.2025

Recap & Final Status

- Kernel Module
- Syscall Table patcher for a single syscall
- Syscall Table patcher with retry for generic syscall
- Python script to generate C code wrapper
- audit.d analysis of syscalls used by server and desktop
- Concept for erorr model creation and decision help for this approach
- Benchmark of single syscall wrap
- Distinction from related work

Recap: Issues with Linux Security Features

- Only functional on Linux 4.X
- 4.X is sufficient to validate the idea
- Current Linux versions use *switch* for syscalls
- cr0 register write-protected, can be easily bypassed with in-line-assembly
- cr register patch, avoid buffer overflow attacks
- *switch* avoid spectre 2 branch prediction attacks (jump into a wrong syscall but not somewhere in kernelspace)
- Race condition in unload

Final Artifact

- Management code around syscall table overwriting for Linux 4.X
- code for code generation
- Concepts and Ideas can be implemented for newer Linux

Final Artifact

```
asmlinkage long syscall_wrapper(struct pt_regs *params) {
  long retval = original_call(params);
  int retry;
  num_used++;

  if (retval < 0) {
    for (retry = 0; retry < NUM_RETRIES; retry++) {
      msleep(retry_intervals[retry]);
      retval = original_call(params);

      if (retval >= 0)
        break;
    }
  }
  return retval;
}
```

Related Work: FIRestarter

- stdlib not syscalls
- Assumes return codes are handled by application
- Assumption some exceptions might not be handled within the application (not rust)
- Converts system library exceptions into handled errors/return codes to trigger different execution path/error mitigation
- Different path for hard/persistent errors
- Soft errors checkpointing and rollback

Related Work: Fault Injection in a High-Performance Computing

- Uses *LD_PRELOAD*
- Assumes that in HPC only specific lib and kernel versions are available, modification possible
- Userspace only

Error Model Concept

- Can cover *all* occurrences of a call
- Analysis/profiling before deciding for relevant calls
- Can help with bad error handling in application code
- Cannot help with hard errors, only transient faults, which are not handled within application code due to missing information
- Error classes distinct return values per syscall

Error Model Concept

- Fault in this case is anything which causes a syscall return code smaller 0 (can be hard or transient fault, see heuristic later)
- Capture error based on fault in syscall
- Prevent error through retry and mitigate fault
- Only escalate error into user mode if fault cannot be mitigated

Heuristic for Error Model Concept

- Heuristics and Syscall retry list use case specific
- For guarded application: Look at syscalls which fail often or rarely?
- Scripts for creating system profile with audit.d for error model creation

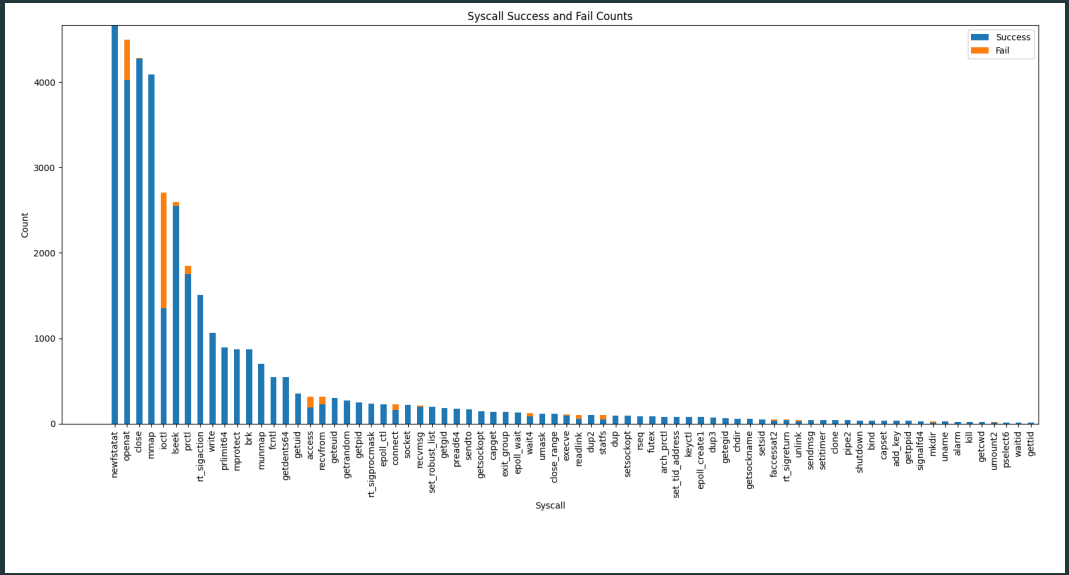
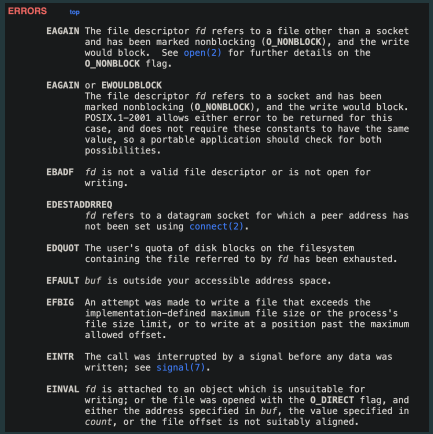


Figure 1: Server Syscall success fail ratio from last presentation

Excerpt from example Error Model



ERRORS top

EAGAIN The file descriptor *fd* refers to a file other than a socket and has been marked nonblocking (**O_NONBLOCK**), and the write would block. See open(2) for further details on the **O_NONBLOCK** flag.

EAGAIN or **EWOULDBLOCK**
The file descriptor *fd* refers to a socket and has been marked nonblocking (**O_NONBLOCK**), and the write would block. POSIX.1-2001 allows either error to be returned for this case, and does not require these constants to have the same value, so a portable application should check for both possibilities.

EBADF *fd* is not a valid file descriptor or is not open for writing.

EDESTADDRREQ
fd refers to a datagram socket for which a peer address has not been set using connect(2).

EDQUOT The user's quota of disk blocks on the filesystem containing the file referred to by *fd* has been exhausted.

EFAULT *buf* is outside your accessible address space.

EFBIG An attempt was made to write a file that exceeds the implementation-defined maximum file size or the process's file size limit, or to write at a position past the maximum allowed offset.

EINTR The call was interrupted by a signal before any data was written; see signal(7).

EINVAL *fd* is attached to an object which is unsuitable for writing; or the file was opened with the **O_DIRECT** flag, and either the address specified in *buf*, the value specified in *count*, or the file offset is not suitably aligned.

- Attached device has sometimes timing/busy issues and syscall fails
- But a retry of *write* could save it from failure

Syscall Retry Matrix for Example

Syscall \ Error	EPIPE	EINVAL	EINTR	ENOSPC
write	y	n	y	y

Table 1: Excerpt from example heuristic for handling broken device write. See: errno.h, errno-base.h and *man* for the syscall.

Handling of persistent and soft errors

- Indication of error type only by return code
- Some error codes may hint at persistent or soft errors
- Mitigation of hard errors can be accelerated through heuristic

Side Effects of Syscalls

- Idempotent syscalls can always be retried
- Non-Idempotent per case decision, see **??**
- *man 2 syscall* description about error code
- For other syscalls retry based on return value
- Related work on check pointing

Failure of Non-idempotent Syscalls

- File system access calls atomic on local File System
- NFS
- openat, unlink, rename ...
- Operation executed but network error

Syscall Retry Matrix

Error / Syscall	EINVAL	ENOSPC	EINTR	ENOMEM	ENOTTY	ENOENT
write	n	y	y	-	-	-
read	n	-	y	-	-	-
mmap	n	-	?	y	-	-
ioctl	n	-	-	-	n	-
open	n	y	y	y	-	y

Table 2: Excerpt only. The Error Codes represent Error Classes, therefore reference possible faults. The table describes a heuristic to prevent a failure. yes/no if a retry should and could help to tolerate the error. - means the error code is not applicable for the syscall. ? would be defined in concrete error Model. See: errno.h, errno-base.h and *man* for the syscall.

Benchmark Setup

- Overhead Measurement in no-fault case
- C program reserves and frees 8092 times 1MiB and calculates Average
- bash script ran program 512 times
- mmap and mem overcommit disabled
- When all calls were wrapped, ioctl often fails therefor often waiting for retries, therefor coose malloc/mmap

Benchmark Expectation

- Slight (5-10%) performance loss
- Assumption: Fault case will have additional waiting time as overhead (not measured)

Benchmarking Results

- ca. 1% overhead
- Average: 147652.044921875 vs 148579.458984375 cycles
- May vary based on syscall usage of the application

How would a production environment look?

- Dynamic: Modify Kernel to allow for hooking into syscalltable
- Static: Modify Kernel directly for retries
- Know your system and create an error Model
- Translate error model into heuristic
- Implement heuristic to work with kernel

Disadvantages

- ca. 1% overhead
- For efficiency a granular analysis is needed
- Also guards syscalls which normally don't fail in specific use cases
- No true black box implementation for efficient usage
- Enforces same behavior for all syscalls

Advantages

- only ca. 1% overhead
- Fault tolerance independent of application code
- Can be modified independent of application
- Can patch application later
- Different approvals for this and application
- Covers *all* syscalls, impossible to miss one

Achieved

- Collected system call statistics of real systems
- Tool to wrap a generic syscall including python script for code generation
- Profiling for creation of error model and related toleration
- Implementation of retry logic in kernel module
- Race condition example to test the retry logic
- Distinction from related work