

www.halfdog.net / [Security](#) / [2013](#) / [Vm86SyscallTaskSwitchKernelPanic](#) /

Share via [f](#) [g+](#)

Introduction

Problem description: The initial observation was, that the linux vm86 syscall, which allows to use the virtual-8086 mode from userspace for emulating of old 8086 software as done with dosemu, was prone to trigger FPU errors. Closer analysis showed, that in general, the handling of the FPU control register and unhandled FPU-exception could trigger CPU-exceptions at unexpected locations, also in ring-0 code. Key player is the *emms* instruction, which will fault when e.g. cr0 has bits set due to unhandled errors.

The exact cause for the observed fault is still not completely clear, there might also be some interaction when running the linux system under test (Debian sid, 3.12-1-486) on problematic CPUs (AMD E-350 Processor)

Methods

[Virtual86SwitchToEmmsFault.c](#) was the first POC, that triggers kernel-panic via vm86 syscall. Depending on task layout and kernel scheduler timing, the program might just cause an OOPS without heavy side-effects on the system. OOPS might happen up to 1min after invocation, depending on the scheduler operation and which of the other tasks are using the FPU. Sometimes it causes recursive page faults, thus locking up the entire machine.

To allow reproducible tests on at least a local machine, the random code execution test tool ([Virtual86RandomCode.c](#)) might be useful. It still uses the vm86-syscall, but executes random code, thus causing the FPU and task schedule to trigger a multitude of faults and to faster lock-up the system. When executed via network, executed random data can be recorded and replayed even when target machine locks up completely. Network test:

```
socat TCP4-LISTEN:1234,reuseaddr=1,fork=1
EXEC:./Virtual86RandomCode,nofork=1
```

```
tee TestInput < /dev/urandom | socat -
TCP4:x.x.x.x:1234 > ProcessedBlocks
```

An improved version allows to bring the FPU into the same state without using the vm86-syscall. The key instruction is *fldcw* (floating point unit load control word). When enabling exceptions in one process just before exit, the task switch of two other processes later on might fail. It seems that due to that failure, the *task->nsproxy* ends up being NULL, thus causing NULL-pointer dereference in *exit_shm* during *do_exit*.

When the NULL-page is mapped, the NULL-dereference could be used to fake a rw-semaphore data structure. In *exit_shm*, the kernel attempts to *down_write* the semaphore, which adds the value 0xffff0001 at a user-controllable location. Since the NULL-dereference does not allow arbitrary reads, the task memory layout is unknown, thus standard change of EUID of running task is not possible. Apart from that, we are in *do_exit*, so we would have to change another task. A suitable target is the *shmem_xattr_handlers* list, which is at an address known from System.map. Usually it contains two valid handlers and a NULL value to terminate the list. As we are lucky, the value after NULL is 1, thus adding 0xffff0001 to the position of the NULL-value plus 2 will turn the NULL into 0x10000 (the first address above *mmap_min_addr*) and the following 1 value into NULL, thus terminating the handler list correctly again.

The code to perform those steps can be found in [FpuStateTaskSwitchShmemXattrHandlersOverwriteWithNullPage.c](#)

The modification of the *shmem_xattr_handlers* list is completely silent (could be a nice data-only backdoor) until someone performs a *getxattr* call on a mounted tempfs. Since such a file-system is mounted by default at */run/shm*, another program can turn this into arbitrary ring-0 code execution. To avoid searching the process list to give EUID=0, an alternative approach was tested. When invoking the *xattr*-handlers, a single integer value write to another static address known from System.map (*modprobe_path*) will change the default *modprobe* userspace helper pathname from */sbin/modprobe* to */tmp//modprobe*. When unknown

executable formats or network protocols are requested, the program `/tmp//modprobe` is executed as root, this demo just adds a script to turn `/bin/dd` into a SUID-binary. `dd` could then be used to modify `libc` to plant another backdoor there. The code to perform those steps can be found in [ManipulatedXattrHandlerForPrivEscalation.c](#).

Results, Discussion

Timeline

- 20131228: Discovery, report at lkml, [full-disclosure](#)
- 20140107: Local-root privilege POC, working both on native CPU and within VirtualBox

Material, References

- Test tool: [Virtual86SwitchToEmmsFault.c](#)
- Random code test tool: [Virtual86RandomCode.c](#)
- Serial console output: [SerialConsoleOutput.txt](#)
- Local-root-excalation: Modify xattr-handler ([FpuStateTaskSwitchShmemXattrHandlersOverwriteWithNullPage.c](#)), execute ring-0 code via xattr-handler ([ManipulatedXattrHandlerForPrivEscalation.c](#))
- Mailing-list reports: [full-disclosure](#)
- Bug reports:
 - Debian bug tracker: [733551](#)

Last modified 20140107
Contact e-mail: me (%) halfdog.net