About

Cross-platform C port of the Copy Fail Linux LPE (CVE-2026-31431). Disclosed 2026-04-29 by Theori / Xint.

#proof-of-concept #exploit #linux-kernel #privilege-escalation #af-alg #local-privilege-escalation #security-research #kernel-exploit #portable-c #nolibc #cve-2026-31431 #copy-fail

Readme

Unknown, MIT licenses found

Activity

40 stars

0 watching

11 forks

Report repository

Releases 3

v0.1.2 Latest

2 hours ago

+ 2 releases

Packages

No packages published

Contributors 1

tgies Tony Gies

Languages

C 96.8% **Makefile** 3.2%

main **1** Branch **3** Tags Go to file Code

tgies feat: /etc/passwd-based exploit f896601 · 8 minutes ago

.github/workflows	feat: /etc/passwd-based exploit	8 minutes ago
nolibc	Initial commit: cross-platform C port of Cop…	8 hours ago
.gitignore	feat: /etc/passwd-based exploit	8 minutes ago
LICENSE-LGPL	Initial commit: cross-platform C port of Cop…	8 hours ago
LICENSE-MIT	Initial commit: cross-platform C port of Cop…	8 hours ago
Makefile	feat: /etc/passwd-based exploit	8 minutes ago
README.md	feat: /etc/passwd-based exploit	8 minutes ago
exploit-passwd.c	feat: /etc/passwd-based exploit	8 minutes ago
exploit.c	Initial commit: cross-platform C port of Cop…	8 hours ago

payload.c	Initial commit: cross-platform C port of Cop…	8 hours ago

README License MIT license

Copy Fail (CVE-2026-31431) - C port

A cross-platform C reimplementation of the Copy Fail Linux LPE (CVE-2026-31431), disclosed 2026-04-29 by Theori / Xint. See the canonical writeup at copy.fail for the full vulnerability description, timeline, and Theori's discovery process.

The publicly-released proof-of-concept is a 732-byte Python script. This C port demonstrates that the same exploit can be expressed as portable C compilable to any architecture nolibc supports, with no per-arch hex blobs or inline assembly in the project's own source.

Author of this port: Tony Gies tony.gies@crashunited.com. Discovery and original disclosure: Theori / Xint.

Repository layout

```
copy-fail-c/
├── exploit.c           the dropper (binary-mutation variant)
├── exploit-passwd.c    the dropper (/etc/passwd UID-flip variant)
├── payload.c           the body that gets dropped (setgid+setuid+execve sh)
├── Makefile            build orchestration
├── nolibc/             vendored from torvalds/linux tools/include/nolibc
└── README.md           this file
```

After `make`:

```
├── payload             tiny static ELF, embedded into the dropper as bytes
├── payload.o           payload wrapped as a relocatable .o by `ld -r -b binary`
├── exploit             dropper, binary-mutation variant
└── exploit-passwd      dropper, /etc/passwd UID-flip variant
```

`exploit.c` opens the target binary read-only, then for each 4-byte window of the embedded payload runs one bogus AEAD-decrypt through AF_ALG whose ciphertext input is supplied via splice() from the target's page-cache pages. The authencesn template's in-place optimization treats the splice'd source pages as both the ciphertext input and the plaintext destination, so the (failing) decrypt has already overwritten the page-cache page by the time authentication verification rejects the request. After 4 * N iterations the target's cached image has been replaced byte-for-byte with the payload. execve()'ing the target loads the mutated pages; the on-disk inode is still setuid root, so the kernel grants root credentials and runs the payload.

`payload.c` is plain portable C: `setgid(0); setuid(0); execve("/bin/sh", ...)`. nolibc supplies the `_start`, the syscall machinery, and the per-arch register-juggling.

A second variant, `exploit-passwd.c`, mutates four bytes of /etc/passwd's page cache instead of a setuid binary's image. It needs no embedded payload and works on systems where the binary-mutation route is blocked, but its cashout surface is much narrower.

Build

Default (host-arch native):

```
make
```

Cross-compile to aarch64 (or any other Linux arch a cross-toolchain is installed for):

```
make CC=aarch64-linux-gnu-gcc LD=aarch64-linux-gnu-ld
```

Architectures supported by the vendored nolibc (per upstream): x86_64, i386, arm, aarch64, riscv32/64, mips, ppc, s390x, loongarch, m68k, sh, sparc. nolibc dispatches on the compiler's arch macros, so picking the right `CC` / `LD` is sufficient.

Required to build:

- a C compiler (`cc` , `gcc` , or any cross variant)
- a linker that supports `ld -r -b binary` (binutils ld and lld both do)
- kernel UAPI headers providing `linux/if_alg.h` and `<asm/unistd.h>` (Debian/Ubuntu: `linux-libc-dev` ; cross variants: typically pulled in by the cross-toolchain package)

There are no external library dependencies. The payload is built freestanding against nolibc; the dropper links against the host libc only for `fprintf` and `perror` .

Architectural choices

Three small toolchain features carry most of the weight in keeping the source portable and the payload small.

nolibc

`nolibc/` is the kernel's tiny header-only libc replacement, vendored from torvalds/linux `tools/include/nolibc/` . It provides `_start` , a portable `syscall()` macro, and inline syscall wrappers, with the per-arch register conventions encoded in `nolibc/arch-*.h` . Building the payload with `-nostdlib -static -ffreestanding -Inolibc` produces a tiny static ELF that calls into the kernel directly without dragging in glibc startup, TLS init, or stack-canary plumbing. Result: ~1.7 KB on x86_64, ~2.0 KB on aarch64, versus ~17 KB for the same `payload.c` linked against musl-static or ~700 KB against glibc-static.

`ld -r -b binary` for embedding

The Makefile turns the built `payload` ELF into `payload.o` via `ld -r -b binary -o payload.o payload` . The linker emits the input bytes verbatim as the data section of a relocatable object file and synthesizes three symbols from the input filename:

```
_binary_payload_start    address of first payload byte
_binary_payload_end      address one past the last payload byte
_binary_payload_size     absolute symbol whose value is the size in bytes
```

`exploit.c` declares the first two as `extern const unsigned char[]` and computes the size as `_binary_payload_end - _binary_payload_start` .

`-Wl,-N` plus tight `max-page-size`

The payload is statically linked with `-Wl,-N -Wl,-z,max-page-size=0x10` , which collapses `.text` / `.rodata` / `.data` into a single LOAD segment with 16-byte file-alignment instead of the kernel-page-aligned 4 KB-per-segment default. This produces an "RWX permissions" warning from `ld` , which is informational only - the payload's runtime memory protection doesn't matter to its single-purpose program. Without this flag, the same code links to ~13 KB on x86_64 (mostly inter-segment zero padding); with it, ~1.7 KB.

Variants and cashout viability

This repository ships two exploit variants that share the AF_ALG/splice page-cache mutation primitive but cash out into root execution differently. Their reliability profiles are not equivalent, and the difference matters when reasoning about real-world threat models.

Binary-mutation variant (`exploit`)

Mutates the page cache of a target setuid binary with the embedded payload bytes, then execs the binary. The kernel grants root credentials from the binary's untouched on-disk setuid bit, loads the corrupted in-memory image, and runs the payload.

Works wherever the attacker can `open(target, O_RDONLY)` for any root-setuid binary on the system. More or less defeated by environments that gate setuid binaries behind restricted-read directories and by setuid-free system designs.

/etc/passwd UID-flip variant (`exploit-passwd`)

Mutates four bytes of /etc/passwd's page cache to set the running user's UID field to "0000". /etc/passwd is world-readable on every standard Linux system, so the *mutation* is universal. Translating it into root execution depends on some root-side process resolving the user via getpwnam/getpwuid and acting on the resolved uid without cross-validation. Many such consumers exist; many of them defensively cross-check against the kernel's view of the calling uid or against on-disk file ownership, breaking the cashout.

Cashout viability matrix

Cashout	Pre-root setup needed	Notes
WSL2 session spawn	No	WSL's per-session `setuid(getpwnam(default_user)->pw_uid)` does no validation. Works cleanly.
util-linux `su`	No	Permissive caller-identity handling.
shadow-utils `su`	Yes	`getpwuid(getuid())` caller-identity check fails because the mutation unmaps the real uid.
sshd (default `StrictModes yes`)	Yes (disable StrictModes)	StrictModes requires the home dir to be owned by root or `pw->pw_uid`. Mutation makes pw_uid=0; on-disk owner stays at original uid; mismatch refuses auth.
MTA local delivery (postfix, exim, etc.)	Variable	Depends on the MDA's home-perm validation. Test per MTA.

Pivoting after `su` fails

`exploit-passwd` execs `su <user>` after mutating, as the simplest possible cashout. That works against util-linux `su` but fails against shadow-utils `su` with "Cannot determine your user name." The page cache mutation is still in place at that point, and pivoting to any other cashout (e.g. using a daemon resolving users via getpwnam without cross-checking) is possible at that point. Run `echo 3 > /proc/sys/vm/drop_caches` as root to clear the corrupted page cache when done testing.

Affected kernels

```
floor:    torvalds/linux 72548b093ee3   August 2017, v4.14
                                        (AF_ALG iov_iter rework that
                                         introduced the file-page write
                                         primitive via splice into the AEAD
                                         scatterlist)

ceiling:  torvalds/linux a664bf3d603d   April 2026, mainline
                                        (reverts the 2017 algif_aead
                                         in-place optimization; separates
                                         source and destination scatterlists
                                         so page-cache pages can no longer
                                         be a writable crypto destination)
```

In between: every major distro kernel that didn't backport the fix. Ubuntu, RHEL, SUSE, Amazon Linux, and Debian were all confirmed vulnerable in their stock cloud-image kernels at disclosure time. Distro-level backports started rolling out around 2026-04-29 alongside the public disclosure. To verify whether a target kernel is in-window, check whether `a664bf3d603d` (or its distro-specific backport) is present in the kernel's git log or the distro's changelog.

License and credits

Discovery and original disclosure of CVE-2026-31431: Theori / Xint. Public writeup: https://copy.fail/.

This C port: Tony Gies tony.gies@crashunited.com

`nolibc/`: vendored from the Linux kernel tree, dual-licensed LGPL-2.1-or-later OR MIT (see `nolibc/nolibc.h` and individual file SPDX headers).

The dropper and payload sources in this repository are released under the same dual LGPL-2.1-or-later OR MIT terms as the nolibc tree they depend on, to keep the licensing trivially compatible for anyone vendoring this whole directory into their own work.

The exploit and payload are published for security-research and defensive-detection purposes. Use against systems you do not own or have explicit authorization to test is your problem, not the author's.