Protocol-flow fuzzing for MPTCP

AI-built, reaching the validation and error-handling code stock fuzzers can't. And the eBPF surface behind it.

We extend BRF (UC Irvine), built on Google's Syzkaller.

mpiric

The harness · github.com/shardulsdk-mpiric/brf

Three names, quickly

THE TARGET

MPTCP

Multipath TCP: one connection running over several network paths at once, say Wi-Fi and mobile data together.

THE BASE FUZZER

Syzkaller

Google's coverage-guided kernel fuzzer, the industry standard. It fuzzes through real syscalls, plus **pseudo-syscalls** for what a plain syscall can't express: helper ops exposed to the fuzzer as if they were syscalls. It chains calls by resource type, but has **no model of protocol state**.

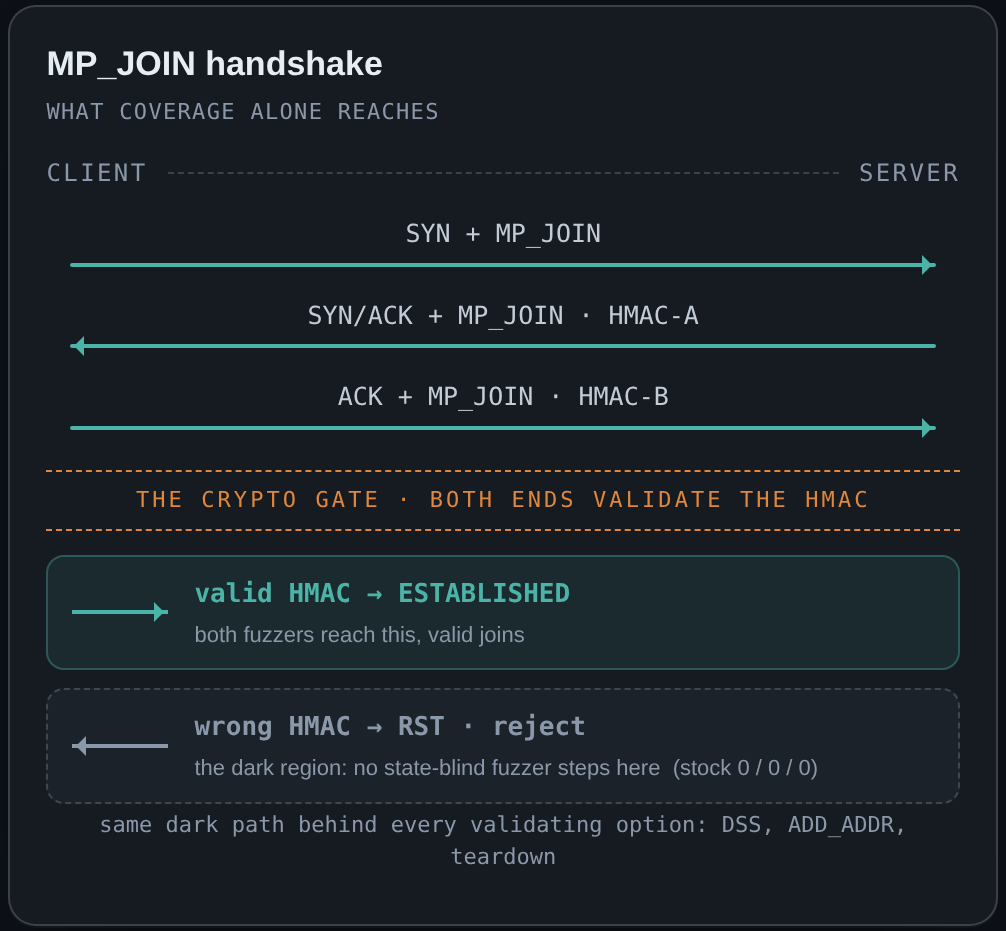
WHAT WE EXTEND

BRF

An academic fuzzer (UC Irvine) that extends Syzkaller with eBPF **pseudo-syscalls**. We copy that pattern for MPTCP: pseudo-syscalls that **carry connection state** across the handshake.

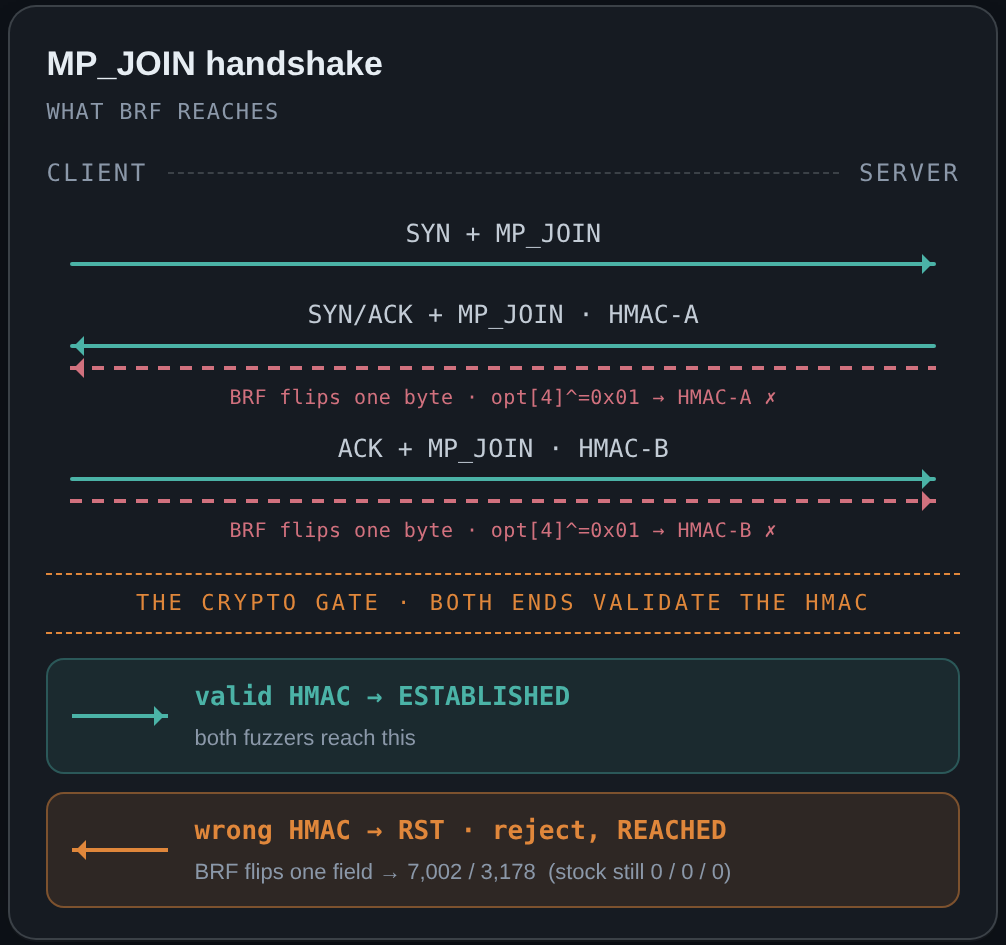
MPTCP adds a path only after a crypto handshake

- A new path is trusted only after a **token**, then **HMACs**. That handshake is a gate.
- The interesting code sits behind it: **validating a join, and rejecting a bad one.**
- Stock Syzkaller **carries no state across the gate**: many valid joins, but no valid-then-wrong input.
- So the reject and error paths stay dark. We measured how dark.



We carry the state, then flip one field

- We own both endpoints over loopback, so we hold the **real token and keys**. The kernel does the crypto; nothing is forged.
- Clear the handshake for real, then **flip one field** on the wire (NFQUEUE), and the dark path lights up.
- One field-flip, every option family: the reject / validation paths become reachable on demand.



What we added to the tool

Three **pseudo-syscall additions** onto the surfaces stock and stock-BRF skip, each mapped 1:1 to a finding. Built with AI, gated by VM-verify.

Stock Syzkaller	+ BRF (UC Irvine)	+ Our MPTCP extension
Coverage-guided fuzzer. Data / options plane and kernel-PM. Carries no state across the MP_JOIN gate.	eBPF runtime fuzzing on Syzkaller: open / load / attach BPF programs. struct_ops left stubbed.	Three additions onto surfaces the two on the left skip.

OUR THREE ADDITIONS, ONE FINDING EACH

1 State-carrier pseudo-syscalls

Own both ends, capture the real token, drive a genuine MP_JOIN, then flip one field.

→ userspace-PM teardown leak + the HMAC-reject / reset paths

2 kcov on the softirq auth paths

Wrap the HMAC validators + option parser so their coverage is visible.

→ the gated auth code becomes measurable

3 struct_ops scheduler generator

Generate / load / fuzz eBPF mptcp_sched_ops (a 0%-covered surface).

→ the BPF kfunc type-confusion, only we reach it

Built on Syzkaller and BRF (Hung + Amiri Sani).

We aim to exceed stock where protocol state is the barrier.

Reaching the gate honestly, not by cracking it

- **Own both ends over loopback**: the harness holds the connection's real token and keys. No cracking, no forging.
- **State-carrier pseudo-syscalls** thread that state call-to-call, via the mptcp_pair resource.
- MP_JOIN is only accepted on a **fully-established** connection, so we complete the DSS round-trip first. A real reachability gotcha.
- NORMAL mode: the gate passes 100%: proof we reach the gated code before mutating anything.

SYZLANG · THE STATE-CARRIER CALLS WE ADDED

```
syz_mptcp_pair_init(...)      /* own both ends */
syz_mptcp_join_subflow(
    pair mptcp_pair, addr_id int8,
    hmac_mut flags[...])
```

WHY IT'S HONEST

The kernel computes every token, nonce and HMAC. We never break crypto; we model the multi-step state machine, then corrupt one field on the wire.

NORMAL fuzzes the flow; MUTATION fuzzes the wire

NORMAL MODE

ordinary fuzzing + reachability proof

Syzkaller still mutates the pseudo-syscall args and sequences here: this is where most of our bugs came from (state machine, teardown, scheduler). The gate also passes 100%, proving we reach the gated code.

MUTATION MODE

the added wire-level layer

NFQUEUE flips one field on egress after the kernel built the real packet: ~29 knobs across 7 option families (join auth, DSS, path-mgmt, teardown). Each is a valid-then-wrong the kernel must reject: the surface stock can never construct.

ISN'T THIS JUST A HAND-WRITTEN HARNESS?

The state SETUP is scripted, yes. But syzkaller fuzzes the flow (NORMAL) and the field values driven into the validator (MUTATION). Both are fuzzing; the near-miss values are exactly what stock can never make valid-then-wrong.

Same interface. One capability added.

- Same Syzkaller dashboard, same corpus view. The sameness is the setup, not the point.
- Watch the gated MIB counters: stock stays 0, ours ticks.
- Each tick is our harness reaching the code that runs only when a credential is invalid.

HONEST CAVEAT

Our VM lacked `/dev/net/tun`, disabling stock's packet-injection route. The claim is "BRF reaches it on demand," not "stock can't."

EXECUTOR · THE ONE-FIELD MUTATION (NFQUEUE WORKER)

```
/* kernel built the real packet; we flip one bit on egress */
opt[4] ^= 0x01;   /* corrupt one byte of the MP_JOIN HMAC */
```

MIB counters

SAME KERNEL · STOCK VS BRF

	STOCK	OURS
MPJoinSynRx join attempts	2,300	51,414
MPJoinAckRx valid joins · both increment	2,245	38,875
HMAC-REJECT / AUTH-FAIL		
MPJoinSynAckHMacFailure SYN/ACK · HMAC-A	0	7,002
MPJoinAckHMacFailure ACK · HMAC-B	0	3,178
BEYOND THE JOIN · DATA INTEGRITY		
DataCsumErr DSS checksum	0	124
MPFailTx DSS-corruption fallback	0	108

3-run baseline. Both arms make valid joins; the count is variable run to run (BRF **10.3k / 13.1k / 15.4k**, stock **17 / 2,228 / 0**).
The conclusive result is the reject path: stock **0** every run vs BRF **10,180**.

Coverage of the MP_JOIN reject path

KCOV · SAME KERNEL, SAME RUNS

covered not covered left = stock · right = BRF

net/mptcp/subflow.c · subflow_syn_recv_sock()

server checks the client's HMAC-B

```
if (!subflow_hmac_valid(subflow_req, &mp_opt)) {
    SUBFLOW_REQ_INC_STATS(req, MPTCP_MIB_JOINACKMAC);
    subflow_add_reset_reason(skb, MPTCP_RST_EMPTCP);
    goto dispose_child;   /* tcp_done() + RST */
}
```

net/mptcp/subflow.c · subflow_finish_connect()

client checks the server's HMAC-A

```
if (!subflow_thmac_valid(subflow)) {
    MPTCP_INC_STATS(net, MPTCP_MIB_JOINSYNACKMAC);
    subflow->reset_reason = MPTCP_RST_EMPTCP;
    goto do_reset;        /* send RST */
}
```

Both cover the check line (valid joins); only BRF covers the reject body.
Reject counter **0/0/0** stock · **2,553/2,437/5,190** BRF.

We drive the reject path on demand

- BRF drives the HMAC-reject/reset paths deterministically: 2,553 / 2,437 / 5,190 per run. Stock 0 / 0 / 0 in our runs.
- **Not** "stock can't", though: our own setup disabled its injection route.
- Where we genuinely lead: struct_ops 0% → reached; userspace-PM 32% → 61%.
- **On the pre-upstream trees.** All of this runs on mptcp/export + export-net, the integration branches maintainers run before mainline, not Linus's tree.
- The data / options plane is stock's strength; ours is the state-gated surfaces it can't construct.

Bugs reported

THE DEMO CRASH

The close-path divide-by-zero

A divide-by-zero in the TCP output path, reached when a userspace-PM address flush tears down a subflow mid-push. One netlink command, a hard oops. On a path stock also fuzzes: "we found it," not "only we could."

EXCLUSIVE TO US

The scheduler kfunc type-confusion

A surface stock can't touch and the academic base never built. Our AI-generated scheduler handed a kfunc the wrong socket subtype; the verifier accepted it; the kernel walked garbage. Genuinely exclusive.

THE CLUSTER

One teardown race, three bugs

The same corner (PM state touched while a connection tears down) gave up three: a leak, its reuse-path sibling, and a NULL deref in the MP_JOIN path. All have fixes on the MPTCP list; the leak was reviewed by sashiko. The deref was in pluggable-PM code only days into net-next. We reach this because we drive the state.

Small N, but each finding opened a neighbouring fix.

The harness is AI-generated

Beyond real syscalls, Syzkaller also supports **pseudo-syscalls**: helper ops exposed to the fuzzer as if they were syscalls. Stock's MPTCP ones keep no state across calls; AI wrote ours to **carry the connection's state** across calls.

STOCK: NO STATE CARRIED ACROSS CALLS

```
socket(AF_INET, SOCK_STREAM, IPPROTO_MPTCP)
sendmsg$MPTCP_PM_CMD_ADD_ADDR(...)  // each call forgets the last
```

OURS (AI-GENERATED SYZLANG): A STATE-CARRIER RESOURCE

```
resource mptcp_pair[intptr]

syz_mptcp_pair_init(...) mptcp_pair   // bring up a pair, do
MP_CAPABLE
syz_mptcp_join_subflow(pair mptcp_pair,
    addr_id int8, hmac_mut flags[...])  // real token; flip a field
```

- The mptcp_pair resource threads the real token call-to-call. That is what lets the fuzzer cross the gate.
- **The workflow:** understand the syzlang idiom, feed the MPTCP protocol (MP_CAPABLE, MP_JOIN, token, HMAC, the state machine), let AI curate a minimal pseudo-syscall set, then refine the executor and verify on real hardware.
- Primarily AI-generated: the syzlang design and the executor scaffolding. Human-refined and hardware-verified.

WHY LEAN ON AI HERE

The bottleneck is holding the syzlang idiom and the protocol at once. AI holds both and produces a working first draft fast, so the human's job becomes review and verification, not authoring from scratch.

Where AI was confidently wrong

- **Silent wrong-destination send.** AI passed the port in network byte order; the kernel byteswaps it again, so every join went to a random port. No crash, no log, no counter.
- **31 false bug reports.** An AI-drafted probe fired on fresh slab memory before we caught that the instrument, not the kernel, was broken.
- **Counter inversion.** AI read the failure counter as the success counter.
- **Self-inflicted.** The fuzzer's own option range hit the harness's own kcov handle, so all 19 "crashes" in a 6h run were the tool scribbling on itself.

THE PROBE THAT FILED 31 FALSE REPORTS (AI-DRAFTED)

```
WARN_ONCE(!list_empty(&anno_list), ...);  // at mptcp_pm_data_init
// INIT_ON_ALLOC=n: raw slab fools list_empty() -> fires every time
```

THE PATTERN

AI-generated descriptions look correct on a code read and fail empirically. The kernel is the only authority on what its own uapi actually does. The failures are not a footnote here, they are the credibility.

Owning the verification is the method

THE VERIFY LOOP

Every AI draft shipped to the corpus only after running on a real kernel and checking four signals agreed with the draft:

- **Coverage:** did it hit the code it claimed?
- **MIB counters:** did the expected event happen?
- **dmesg:** any warning it shouldn't produce?
- **Selftest:** does the protocol behave end to end?

WHY ALL FOUR

Each signal catches a class the others miss. The byte-order bug passed dmesg, no crash, but failed coverage and counters, a silent no-op.

"No crash" is blind to exactly the dangerous failures, so we never accept it as success.

Our working principles

Developed through the work and articulated with AI's help. Its confidence carries no signal, so we treat its output like a patch from an unproven contributor: each rule below replaces trust with a check.

- **Anchor to ground truth,** not the AI's word. The absence of a red flag is not a green flag; require positive evidence.
- **Involvement scales inversely with abstraction distance.** Hands-on in your domain; delegate outside it, but keep the outcome check.
- **Go as deep as the reasoning holds.** Where it stops holding up, dig a level deeper and re-sync with the AI rather than taking over.
- **A correction isn't done** until the lesson is institutionalized into the AI's task context.
- **Separate durable rules from disposable.** The verification scaffolding outlives any model version; a model's quirks decay, so re-run the calibration each release, not once.
- **Quarantine unverified output** at the boundary, so one wrong step can't poison what's built on it.

> THE GUARDRAIL
>
> Never abstract past a checkpoint you can verify. That is what keeps "delegate outside your domain" from decaying into blind trust.

A reusable recipe, not a one-off

HONEST LIMITS

- Small N; the baseline is a fair comparison, not a claim stock is bad.
- Raw data available on request; the scheduler bug is privileged-local.

THE MENU

- **In MPTCP:** wire-level option mutation, kernel-PM mode, the reset surface.
- **The eBPF frontier:** the same generator points at bpf_tcp_ca and other struct_ops surfaces.
- **Cross-protocol (hypothesis):** in-kernel QUIC, the TLS handshake path.

Every new surface we tried produced a bug. Come build on it.

Reach the code behind the gate. Build it with AI.

? What surface would you point this at next?

? How are you gating what the AI drafts before you trust it?

Mpiric Software · extends **BRF** (Hung + Amiri Sani, arXiv:2305.08782)
built on **Syzkaller** (Dmitry Vyukov et al.)