MPTCPcap: A Practical Framework for Security Testing of Multipath TCP

Riccardo Mancini[1], Federica Bianchi[1,*] and Angelo Spognardi[1]

[1]Computer Science Department, Sapienza University of Rome, Italy

Abstract

Multipath TCP (MPTCP) improves reliability and performance by enabling multiple network paths for a single connection, but its security remains a concern. In this paper we present `MPTCPcap`, a real-time and reusable analysis tool for monitoring MPTCP connections and simulating attacks. The tool tracks subflows, endpoints, and keys in real time and supports packet injection, modification, and dropping, enabling emulation of off-path, partial on-path, and full on-path adversaries. Using `MPTCPcap`, we reassess well-known MPTCPv0 attacks against the Linux MPTCPv1 implementation and identify two previously undocumented vulnerabilities: (i) an off-path attack that closes subflows in Linux's MPTCP implementation, and (ii) a protocol-level weakness allowing an attacker with connection keys to prevent the creation of new subflows. Our results highlight how adversarial testing of MPTCP implementations can expose residual security issues and provide a practical basis for future security analysis and mitigation efforts.

Keywords

Multipath TCP, MPTCP Vulnerabilities, MPTCP Testing Framework, Network Security

1. Introduction

The Multipath Transmission Control Protocol (MPTCP) was introduced as an extension of TCP in RFC 6824 [1] to enable the simultaneous use of multiple paths within a single connection, improving its throughput and reliability. At its core, MPTCP splits a single logical connection into multiple TCP subflows, each following a distinct path. It operates at the transport layer while remaining fully compatible and transparent to higher-layer applications and lower-layer protocols. When a peer does not support MPTCP, the connection seamlessly falls back to standard TCP, allowing backward compatibility and easing deployment in existing infrastructures.

MPTCP adoption has grown steadily over the past decade, primarily driven by Apple, which employs it in services such as Siri, Maps, and Music to improve reliability and reduce latency[1]. It has also been integrated into the Linux kernel since version 5.6, and its use continues to expand: between 2021 and 2022, the number of MPTCP-capable IPs doubled compared to the previous year [2], with tens of thousands of servers currently supporting the protocol [3]. These trends highlight that MPTCP is now part of real-world, large-scale deployments.

Although MPTCP increases reliability, it also introduces new attack surfaces. Since MPTCP uses multiple paths, a natural consequence is that it opens many doors for an adversary to gain access to its session, making possible attacks such as man-in-the-middle (MITM), denial-of-service (DoS), and SYN-flooding attacks [4]. Indeed, like TCP, MPTCP does not include authentication or integrity protection by default, relying instead on upper-layer security protocols such as TLS. Consequently, when MPTCP is deployed without additional security layers, it remains vulnerable to attacks that exploit its lack of cryptographic protection.

Several studies have analyzed the security of the protocol, exposing attacks such as subflow hijacking, traffic diversion, and address advertisement abuse. In response, the original specification (MPTCPv0)

Joint National Conference on Cybersecurity (ITASEC & SERICS 2026), February 09-13, 2026, Cagliari, IT

*Corresponding author.

riccardo.mancini01@gmail.com (R. Mancini); bianchi@di.uniroma1.it (F. Bianchi); spognardi@di.uniroma1.it (A. Spognardi)

0009–0006–2698–4484 (F. Bianchi); 0000–0001–6935–0701 (A. Spognardi)

[1]https://support.apple.com/en-us/101905

was revised to MPTCPv1, introducing stronger keying and authentication for several control operations. However, even with these updates, residual threats persist because many control messages remain only partially authenticated, and real implementations may diverge from the standard. As a result, evaluating the robustness of deployed MPTCP stacks remains a practical challenge.

While MPTCP can rely on TLS to protect application payloads, several dedicated approaches have also been proposed to enhance transport-layer security, such as Multipath TLS (MPTLS) [5] and Secure MPTCP [6]. Nevertheless, these extensions are not widely adopted, and attacks remain feasible during subflow negotiation, before encryption begins, or in deployments that omit TLS entirely.

Despite extensive research on MPTCP's design and vulnerabilities, existing tools focus primarily on performance monitoring rather than vulnerability assessment. Tools such as `mptcptrace` [7], `mptcplot` [8], and `mptcpanalyzer` [9] are valuable for passive analysis and measure metrics like throughput, scheduler behavior and reinjection rates. The only tool allowing controlled packet injection, Packetdrill [10], is designed for functional validation rather than dynamic attack simulation. Consequently, there is still no practical framework for systematically testing the security of MPTCP implementations or assess of protocol behavior under adversarial conditions.

To fill this gap, we developed `MPTCPcap` a real-time packet capture and attack simulation tool for empirical MPTCP security assessment. The tool monitors and analyzes MPTCP connections in real time, tracks subflows and hosts' endpoints, and allows simulated attacks by controlling the injection, modification, or dropping of packets according to configurable attacker models (off-path, on-path, or partial on-path). For each captured packet, the tool also logs essential metadata, including its source and destination addresses, sequence and acknowledgment numbers and MPTCP option subtype. This architecture allows reproducible and fine-grained analysis of both protocol and implementation behavior under adversarial conditions.

Using `MPTCPcap` we successfully reproduced several known MPTCP attacks, including key eavesdropping, traffic diversion via Change Priority and unauthorized subflow creation. More importantly, the tool allowed us to identify two previously undocumented vulnerabilities: (i) a flaw in the Linux MPTCP implementation that allows an off-path attacker to arbitrarily close active subflows using forged Address Removal messages, and (ii) a protocol specification weakness that enables an attacker possessing the MPTCP connection keys to prevent the hosts from establishing new subflows, effectively exhausting subflow slots.

The discovered issues, therefore, remain relevant in real-world scenarios where MPTCP operates without TLS, or where subflow negotiation occurs before encryption. Our testing framework aims to complement ongoing work on Multipath TLS and Secure MPTCP [5, 6] by providing a practical foundation to assess residual vulnerabilities and implementation robustness.

In summary, we make the following contributions:

- Development of `MPTCPcap` a real-time MPTCP security analysis tool, capable of performing connection analysis and attack simulation. The tool offers a practical and reproducible framework for systematic MPTCP security testing and vulnerability assessment;
- Discovery of a new vulnerability in Linux MPTCP implementation, which allows an off-path attacker to close subflows via Address Removal messages;
- Discovery of a vulnerability in the protocol specification that allows an attacker with connection keys to prevent legitimate hosts from establishing new subflows.

The rest of the paper is organized as follows. Section 2 reviews prior research on MPTCP security and analysis tools. Section 3 provides background on MPTCP. Section 4 presents the design and implementation of our tool. Lastly, Section 5 discusses the discovered vulnerabilities.

2. Related works

Research on MPTCP security has evolved along two primary directions: (1) the study of protocol-level vulnerabilities and mitigation strategies, and (2) the development of tools for analyzing, testing, or monitoring MPTCP implementations.

Vulnerability Analysis. The security of MPTCP has been an ongoing concern since its early specification. Bagnulo et al. [11] analyze MPTCP threats, highlighting the lack of authentication in the ADD_ADDR option of MPTCPv0, which enables session hijacking and man-in-the-middle attacks. They also described denial-of-service and flooding amplification attacks exploiting the MP_JOIN handshake and discussed the risk of key exposure through passive eavesdropping. Demaria [12] reproduces the Address Advertisement vulnerability identified in [11], and mitigates it by authenticating the contents of the subtype with a Hash-based Message Authentication Code. Munir et al. [13] discuss two MPTCP vulnerabilities that enable traffic diversion and connection hijacking. One of them, originally described by Shafiq et al. [14], allows an attacker to infer the throughput of other subflows by eavesdropping on one subflow and analyzing sequence numbers and data sequence numbers in data-sequence-signaling (DSS) subtypes. Shafiq et al. also exploited the MP_PRIO to perform a hijacking attack by setting all subflows as backup except the one under the attacker's control, so that he can intercept all the exchanged packets. Cao et al. [15] analyzed MPTCP's robustness under cyberattacks incomplete network knowledge, modeling selective and random subflow disruptions through graph-based simulations. Kumar et al. [16] analyzed manipulation of DSS subtype to trigger flooding and congestion, leading to bandwidth exhaustion and buffer overflow. Shafique et al. [17] analyzed the same threats discussed in [11] and evaluated the effectiveness of encryption-based versus non-encryption-based defenses. Meira et al. [18] studied how MPTCP's multipath nature can bypass network intrusion detection systems by distributing malicious payloads across independent subflows. A review of multipath transport protocols by [19] highlights that MPTCP communication introduces additional security requirements, such as secure handshaking, authenticated addition and removal of subflows, and resilience against flooding and hijacking attacks, warning that these mechanisms are still not fully addressed in current implementations.

Testing frameworks and analysis tools. In parallel, a variety of tools have been created to analyze MPTCP behavior, yet almost all target performance evaluation or functional correctness rather than security testing.

The `mptcptrace` tool [7] extends the packet trace analyzer `tcptrace`[2]. While `tcptrace` provides metrics for TCP connections by analyzing packet traces, it cannot link different TCP streams belonging to the same MPTCP connection. The `mptcptrace` tool overcomes this limitation and provides information about the entire connection, such as *round-trip times* for each subflow, sequence and acknowledgment numbers analysis, packet size, and packet retransmission rate over different subflows. The `mptcplot` tool [8] also works with packet captures, providing information such as connection level sequence and acknowledgment numbers, throughput, source and destination addresses used in MPTCP connections and their subflows. The `mptcpanalyzer` tool [9] focuses on the analysis of packet re-injection (recognizing necessary and unnecessary re-injections) and on measuring the connection's goodput to assess the performance of the scheduling algorithm used to decide on which subflow each packet will be sent. The tool also provides details about packet count, payload size, keys and tokens used by the connection and acknowledgment numbers. `packetdrill` [10] was developed to ensure correct network stack implementations through the definition of test cases. The tool allows writing scripts that open network sockets and send network packets. The user can define packets to be sent at specific timestamps, simulating one of the hosts, and the packets that should be received as a response.

Beyond trace-analysis tools, Linux-centric testing and control utilities, such as `mptcp-tools` [20], `mptcpize` [21] (transparent MPTCP socket creation), `ip mptcp` [22] (in-kernel path-manager configuration), and `mptcpd` [23] (userspace path manager), are used to run and manage MPTCP experiments, but they target configuration and path-management rather than adversarial or security testing.

Thus, no existing tool offers a reusable and systematic way to empirically test the security of live MPTCP implementations, leaving a gap that this work addresses.

3. Preliminaries

A Multipath TCP connection [24] extends standard TCP by allowing multiple concurrent TCP streams, called subflows, between the same pair of hosts. It operates entirely at the transport layer and remains transparent to applications and lower network layers: to the application it appears as a single TCP connection, while each subflow is a normal TCP stream at the IP level.

MPTCP distinguishes itself from TCP through a dedicated TCP option type that carries control information used to coordinate subflows. Each option uses a specific subtype to perform control operations such as establishing new subflows, advertising or removing addresses, or terminating the connection. Each subflow is represented by the standard TCP four-tuple of source and destination IP addresses and port numbers. Each pair is referred to as *address*.

Figure 1 illustrates how MPTCP establishes multiple subflows between two hosts, each equipped with two interfaces (A1, A2 and B1, B2).

1. The first subflow between A1 and B1 is created with a standard TCP handshake that includes the `MP_CAPABLE` subtype option, enabling MPTCP and exchanging keys for authentication;
2. The client can then initiate another TCP handshake using `MP_JOIN` (e.g., from A2 to B1) to attach a new subflow to the same connection;
3. Additional reachable addresses may be announced using `ADD_ADDR`, enabling further subflows such as A1–B2;
4. Subflows can later be closed individually with `REMOVE_ADDR` or a normal TCP termination, without affecting the overall connection.

We will now discuss some of the MPTCP option subtypes, which will be used in the attacks described in Section 5.

[2] http://www.tcptrace.org/

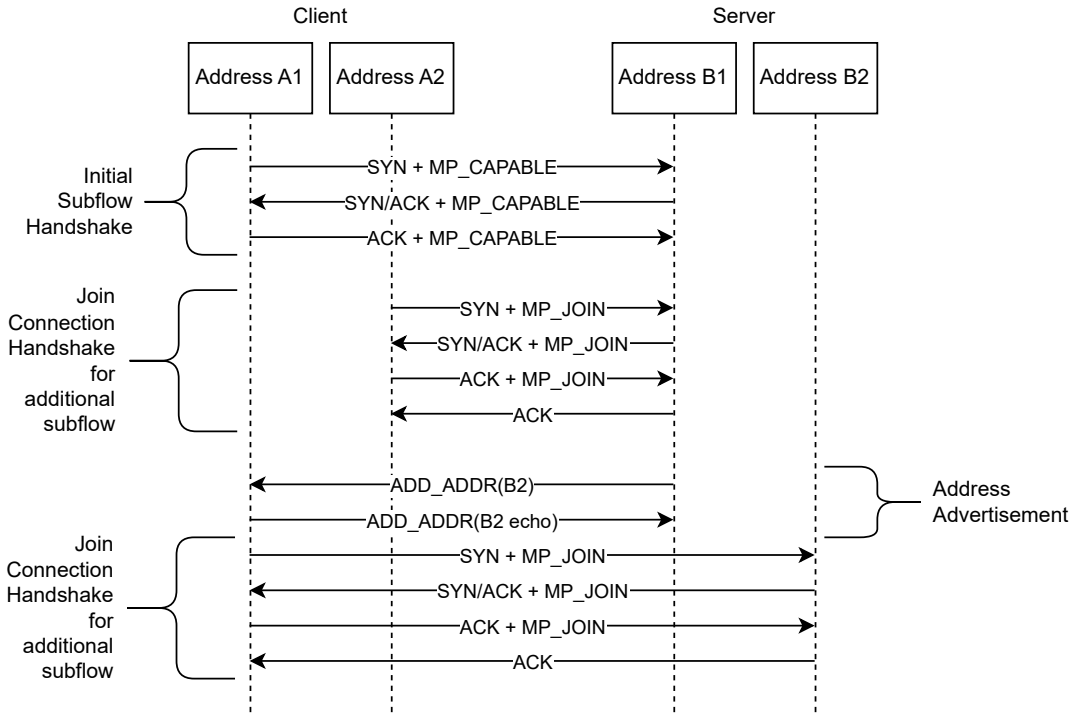

Figure 1: Example of subflow creation between host A and B, both with two addresses.

Initial handshake with `MP_CAPABLE`. The `MP_CAPABLE` subtype appears during the first TCP handshake, indicating that the sender supports MPTCP and wishes to enable it for the connection. Additionally, this subtype enables the exchange of keys and the negotiation of connection preferences. It enables both hosts to exchange 64-bit keys in plaintext, later used to: (i) derive initial connection-level sequence numbers; (ii) compute HMACs authenticating control packets (e.g. `ADD_ADDR`); (iii) generate tokens used in `MP_JOIN` to identify the connection.

New subflow initiation with `MP_JOIN`. After the initial subflow is established, additional ones can be added using the `MP_JOIN` option subtype during a normal TCP handshake. The initiator includes its Address ID, the peer's Token, and a random nonce.

The Token is a cryptographic hash derived from the receiver's key, used to identify the MPTCP connection to which the new subflow belongs. To prevent unauthorized actors from creating new subflows, both peers validate HMAC values derived from their exchanged keys. Packets with invalid HMACs or unknown Tokens are rejected with TCP RST.

Address removal with `REMOVE_ADDR`. When interfaces become unavailable, a host sends `REMOVE_ADDR` to inform its peer that one or more addresses are no longer valid, allowing it to close the associated subflows. The subtype includes one or more Address IDs corresponding to the sender's addresses that need to be removed. Because `REMOVE_ADDR` may be sent on any active subflow, receivers should confirm the affected paths (e.g., via a keep-alive probe) before closing them. While intended for address management, misuse of this message can prematurely terminate active subflows, one of the vulnerabilities explored in Section 5.

Connection termination with `MP_FASTCLOSE`. In regular TCP, a connection can be abruptly closed by sending a RST signal. In MPTCP, each RST affects a single subflow, thus to terminate the entire multipath session, hosts use `MP_FASTCLOSE`.

The sender transmits this option containing the peer's key and then issues RSTs on all subflows. Upon receipt, the peer mirrors the process, ensuring consistent teardown of all subflows.

4. `MPTCPcap`: A Novel Tool for MPTCP Security Assessment

As seen in Section 2, existing MPTCP tools focus on performance measurement, but they provide no means to simulate attacks or analyze protocol behavior under adversarial conditions.

To close this gap, we developed `MPTCPcap` a real-time packet capture and attack-simulation tool that allows researchers to observe and manipulate MPTCP traffic during live connections. The tool provides a controlled environment where known and novel attacks can be simulated, packets can be intercepted or forged, and the effects of each attack can be measured at both the protocol and connection level. It combines active manipulation of live MPTCP traffic with real-time monitoring and a minimal but extensible architecture.

The main design goals of `MPTCPcap` are:

1. **Clarity:** display only information relevant to the security analysis (subflows, option subtypes, sequence/ack numbers) instead of overwhelming packet-level details.
2. **Extensibility:** allow the rapid implementation of new attack modules without modifying the core engine.
3. **Reusability:** provide a flexible environment that can be deployed in different network topologies or experimental setups.
4. **Reproducibility:** make experiments easy to rerun and compare (same inputs, same placement, same logs).

4.1. System Architecture and Components

The MPTCPcap framework consists of three coordinated software components:

- *MPTCP client*: initiates MPTCP connections and generates application traffic.
- *MPTCP server*: accepts the client's connection, replies with confirmation messages, and provides the symmetric control traffic needed to trigger protocol mechanisms.
- MPTCPcap *tool*: positioned between the two endpoints, intercepts packets, analyzes MPTCP options, modifies or forges packets when required, and logs all relevant state information.

All components are implemented in the Go programming language for its concise syntax and built-in support for MPTCP since version 1.21. The gopacket and go-nfqueue libraries provide, respectively, packet parsing/crafting and bindings to the Linux Netfilter framework. This makes the environment lightweight, portable, and easy to extend.

The client and server are intentionally minimal to keep the resulting traffic clear and interpretable. The client sends short messages to the server, which replies with confirmation messages. This exchange naturally produces the MPTCP control subtypes needed for testing. Under normal operation:

- The client sends a message on a subflow selected by its scheduler;
- The server acknowledges each packet on the same subflow;
- After receiving the full message, the server replies with a confirmation message;
- The client acknowledges the response.

The client also includes a mode that allows for the constant sending of messages, which helps generate high throughput that can then be measured over time on each subflow.

Topology and placement MPTCPcap is designed to operate on a node that can observe and manipulate all subflows between two communicating hosts, and thus manipulate the packets exchanged between them. Figure 2 shows an example in which both hosts have two network interfaces, and the host that is running MPTCPcap is placed on a node through which all subflows created by the two hosts will be forced to go.

This guarantees that MPTCPcap can observe all subflows without modifying the endpoints themselves, which is essential for realistic and repeatable testing. The Linux Netfilter subsystem handles interception: a launcher script dynamically installs iptables rules that forward packets to a user-space queue through NFQUEUE, allowing MPTCPcap to decide for each packet whether to accept, modify, or drop it.

However, placement determines which attacker classes can be emulated: a node on all subflows can reproduce full on-path attackers; if the tool can observe only a subset of subflows, it can emulate partial-time on-path attackers (temporary control of subflows); the framework can also be configured to attempt off-path behaviors (spoofing/injection) when keys or sequence information are available.

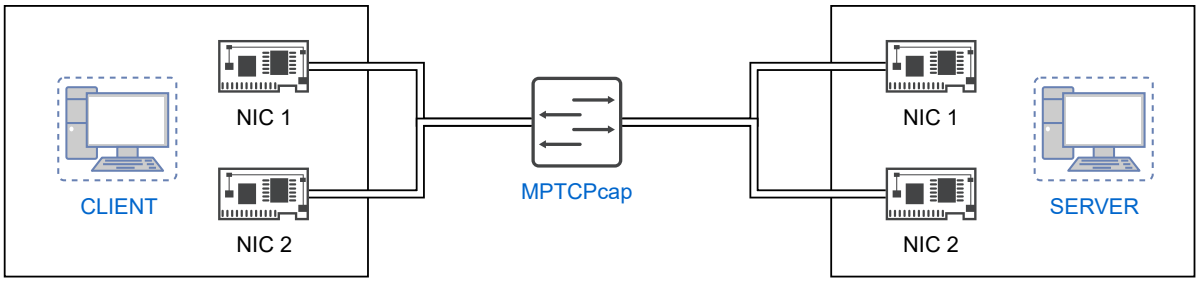

Figure 2: MPTCPcap should be run on a machine that is on the path of all subflows.

4.2. MPTCPcap Capabilities and Functional Overview

Monitoring and logging. MPTCPcap continuously reconstructs and maintains the full MPTCP connection state: it records both hosts and their endpoints (IP/port tuples and Address IDs), the set of active and pending subflows, per-flow sequence numbers and the keys observed during the MP_CAPABLE handshake. This is necessary to support subsequent attacks that require that state.

The tool monitors all packets delivered to it but processes only those that contain the MPTCP TCP option; packets that lack the option are returned unchanged unless explicitly targeted by an active attack module. For every MPTCP packet, the framework emits a compact, structured log entry that includes the packet's source and destination, the Sequence and Acknowledgement numbers, the MPTCP option subtype, and the action taken by the attacker-simulation layer (accepted, modified, dropped, or created/forged by the attacker). This compact logging format intentionally trades raw pcap verbosity for reproducible, analyzable traces that contain exactly the information required to verify whether an attack succeeded or to re-run an experiment deterministically.

Attack simulation and modularity. The tool is designed to simulate adversaries and to run a broad spectrum of attack scenarios. It is capable of extracting the two hosts' keys from observed MP_CAPABLE exchanges and of tracking per-subflow sequence and acknowledgement progression. Using this state, MPTCPcap can determine which injected or forged control messages would be accepted by a given implementation.

The framework determines the subflow of each packet by matching its four-tuple and observing TCP handshakes; it also identifies new subflows by analyzing MP_JOIN and related handshake messages. Subflows that are created by the simulated attacker are tracked in the same internal model, so the attacker module is always aware of exactly which subflows it controls and which ones remain controlled by the legitimate endpoints.

Attack logic is implemented as modular handlers: each handler receives the packet context and the connection model and returns a decision (accept/drop/modify) and optionally requests forged packet emission. This modular API keeps new-attack code minimal and easy to implement.

Interaction and workflow. To make experiments practical and repeatable, MPTCPcap exposes both an interactive text-based user interface (TUI) and a scriptable CLI. The TUI allows experimenters to inspect the current connection model (subflows, endpoints, Address IDs, and observed keys) and to enable, disable, or cancel attacks at runtime without restarting the tool or the MPTCP connection. This interactive capability is important because restarting the monitor would require reestablishing subflows and re-exchanging keys, disrupting reproducibility. The TUI therefore enables sequences of attacks and observations within a single continuous connection. For unattended or batch experiments, the CLI and automation scripts reproduce the exact same actions so that experiments can be run repeatedly and logs collected deterministically. Throughout interactive and automated runs, the tool records the compact logs described above and can optionally produce parallel PCAP/Wireshark traces for low-level verification.

Capture and interception concepts. The implementation of MPTCPcap focuses on providing a lightweight and extensible framework capable of real-time packet interception, modification, and injection, without requiring changes to the endpoints under test.

To support real-time manipulation of active MPTCP connections, the tool relies on the standard Linux packet-filtering pipeline to deliver selected traffic to user space (using NFQUEUE). The interception host already forwards packets between client and server; enabling the tool simply configures the system so that these forwarded packets are made available to MPTCPcap before being reinjected into the network stack. This avoids modifying either endpoint and keeps deployment lightweight: any router, bridge, virtual switch, or VPN gateway can be turned into an adversarial vantage point by running the tool. In user space, incoming packets are decoded using the gopacket library, while custom serializers and deserializers handle all MPTCP option subtypes used in the experiments. Outgoing forged packets are

transmitted via raw sockets to ensure full control over headers, including the ability to spoof source addresses, and to prevent tool-generated packets from re-entering the interception chain. This split between kernel-level interception and user-space logic keeps the tool portable and modular while allowing precise manipulation of MPTCP control traffic.

4.2.1. Packet handling workflow

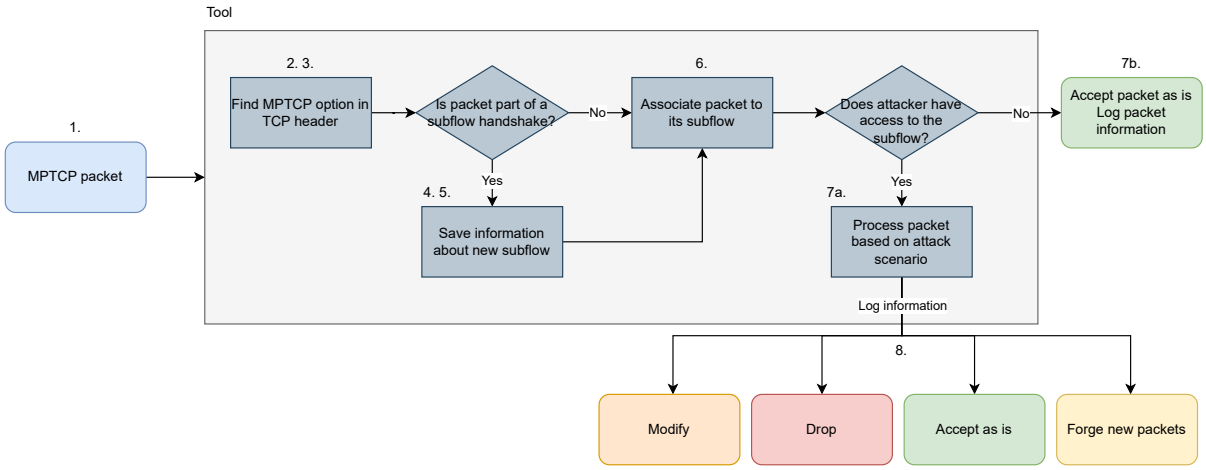

Figure 3: How each packet is handled by `MPTCPcap`.

Below is a high-level explanation of how each packet is processed, as shown also in Figure 3:

1. The packet bytes are received from the queue.
2. The presence of an MPTCP option is checked. If the packet does not contain one, it does not belong to the MPTCP connection and is then accepted. Otherwise, the packet is handled by `MPTCPcap`.
3. The MPTCP option is parsed to identify its subtype and extract its associated information.
4. If the subtype is `MP_CAPABLE`, the keys contained within it are stored for potential use by the attacker. Information about the initial subflow is also saved, and the source and destination IP addresses and TCP ports are associated with the two hosts.
5. If the subtype is a `MP_JOIN` instead, information about the new subflow is saved. If the new subflow has a new endpoint, it is added to its host's endpoints list.
6. The packet is associated with the corresponding subflow looking at the 4-tuple.
7. If the packet belongs to a subflow under the attacker's control, it is processed according to the simulated attack. Otherwise, it is accepted.
8. Depending on the outcome of the packet processing, the packet is either dropped, accepted, or modified and then accepted.

4.2.2. Attack Processing

Attacks in `MPTCPcap` are implemented as small, independent modules, that expose a single handler function invoked whenever the tool intercepts a packet on a subflow the attacker is assumed to control. The handler receives the decoded packet together with the current connection state and returns a verdict to accept, drop or forward a modified version. The handler may also request the transmission of additional raw packets, enabling the simulation of various attacks. Forged packets are sent using raw sockets, giving attack modules full control over all header fields, including spoofed source IPs and ports, and ensuring that injected packets bypass the interception chain.

The ease of implementing new attacks comes from the fact that the module only describes what to do with each packet, while the framework handles everything else: packet interception via Netfilter, MPTCP option parsing and serialization, subflow identification, connection-state updates and raw-socket emission. Because all low-level protocol details are handled by shared helper libraries, a new attack typically requires only a few dozen lines of logic.

The following minimal example illustrates a module that sends a forged REMOVE_ADDR after observing a DSS packet on a controlled subflow.

Listing 1: Handler implementation of forged REMOVE_ADDR.

```
type RemoveAddrAttack struct {
   done  bool
   addrIDs []uint8
}

func (a *RemoveAddrAttack) Handle(pkt *mptcp.Packet) {
   if a.done || pkt.Subtype != mptcp.DSS {
      return
   }

   opt := mptcp.Sub_REMOVE_ADDR{ AddrIDs: a.addrIDs }
   forged := BuildFrom(pkt, opt) // reuse IP/TCP headers and replace MPTCP option
   SendPacket(forged)
   LogCreatedPacket(forged)

   a.done = true
}
```

5. New Vulnerabilities Discovered with MPTCPcap

This section revisits known MPTCP weaknesses and evaluates them against the current MPTCPv1 Linux implementation using our testing framework. We also present two previously undocumented vulnerabilities exposed by our tool. This shows how a reusable, controllable testbed such as MPTCPcap helps reproduce, adapt and systematically stress specific protocol behaviours.

Experimental Setup. All experiments were performed on a controlled three-node testbed designed to (i) expose the full MPTCP control plane to the attacker; (ii) reliably generate multiple subflows; (iii) allow packet interception and modification without altering the endpoints.

The **client** runs inside a lightweight QEMU virtual machine. The VM is configured with multiple virtual network interfaces. The VM executes a Linux kernel with MPTCPv1 support and a simple round-robin MPTCP scheduler, ensuring that all established subflows actively carry traffic. This makes throughput shifts caused by attacks immediately observable. The host machine that runs the VM also runs MPTCPcap which simulates the attacker. All VM interfaces are attached to a Linux software bridge on the host. The bridge forwards packets using NFQUEUE, which delivers selected traffic to MPTCPcap before reinjecting it into the network stack. As a result, the attacker acts as a transparent Layer-2 switch positioned in the path, guaranteeing that every packet exchanged between client and server traverses the MPTCPcap tool. The **server** is a remote Linux machine (kernel 6.12 in our tests) configured with two independently reachable IP addresses, enabling creation of multiple subflows from the client.

The final network topology is shown in Figure 4.

5.1. Threat Model

The impact and feasibility of each attack depend primarily on the adversary's vantage point with respect to the MPTCP connection. Following the taxonomy in [11], we consider three attacker classes:

- **Off-path attacker.** The attacker is not located on any of the paths used by the MPTCP connection. She cannot see live packets, but may know the IP addresses and ports of the endpoints and can inject spoofed traffic.

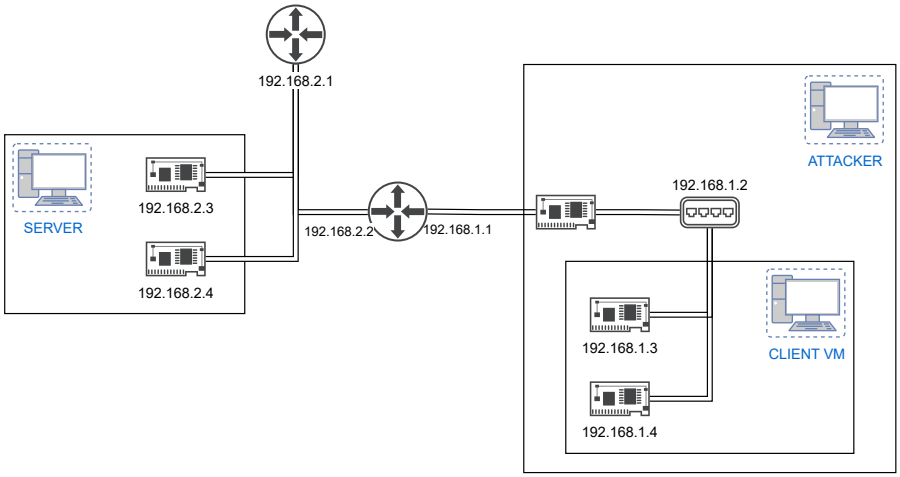

Figure 4: Network topology used for tests.

- **Partial on-path attacker.** The attacker is on the path of at least one subflow for a limited time or for the whole connection, but cannot observe all subflows. This may correspond, for example, to a router traversed by only one of several subflows.
- **Full on-path attacker.** The attacker can intercept and modify packets on all subflows for the entire lifetime of the connection, effectively acting as a man-in-the-middle.

5.2. Reassessment of Known Vulnerabilities

From the literature analyzed in Section 2, we reassessed the following vulnerabilities in MPTCPv0: (i) Key Eavesdropping; (ii) Traffic Diversion via `MP_PRIO`; (iii) Subflow Creation via `ADD_ADDR`.

We re-implemented the attack logic within `MPTCPcap` and replayed it against the Linux MPTCPv1 implementation. This allowed us to verify whether the protocol behaviour matched what the RFC update intended to fix.

5.2.1. Key Eavesdropping

This vulnerability was discussed in [11], which acknowledges that `MP_CAPABLE` exposes the two host keys in cleartext. The initiator's key appears in the third packet of the three-way handshake, while the responder's key appears in the SYN/ACK. Thus, an attacker can eavesdrop on the keys exchanged during connection establishment [11]. Version 1 no longer includes the initiator's key in the SYN as in v0, but the vulnerability persists, since the protocol still transmits both keys in plaintext in the SYN/ACK and final ACK of `MP_CAPABLE`.

Since this vulnerability requires a partial on-path attacker, `MPTCPcap` was placed on the first subflow and run in passive mode. Because the client VM's traffic is bridged through the attacker node in our topology, the tool can observe the unmodified `MP_CAPABLE` handshake. During reassessment, we confirmed that the SYN/ACK carries the responder's key and the final ACK carries both keys; `MPTCPcap` automatically extracted these fields as expected.

This behaviour is RFC-compliant and considered an acceptable design trade-off, but it remains the enabling step for all key-dependent attacks presented later, including fake subflow creation and subflow-slot saturation.

5.2.2. Traffic Diversion via MP_PRIO

In MPTCPv0, the MP_PRIO option allowed an endpoint to mark all subflows associated with a given Address ID as backup or non-backup. Since the subtype included the Address ID explicitly, a single forged MP_PRIO packet was sufficient for an attacker to change the priority of every subflow tied to that address. As shown by Munir et al. [13], this enabled two attacks: (i) connection hijacking, where an on-path attacker controlling one subflow forces all other subflows into backup mode so that all traffic flows through the compromised one; (ii) directed traffic diversion, where the attacker redirects throughput toward or away from specific paths. To prevent these diversions, MPTCPv1 modifies MP_PRIO so that it no longer contains an Address ID and only affects the subflow on which it is sent. In the updated specification, an attacker must inject one valid forged packet per subflow, each carrying the correct TCP sequence number.

To reassess the vulnerability, using MPTCPcap we implemented the v1 MP_PRIO format and added the ability to inject forged priority-change packets on arbitrarily chosen subflows. For each subflow, the tool forged a valid MP_PRIO (with spoofed sequence numbers) and logged: (i) whether the packet was accepted by the host, (ii) whether the subflow's backup flag changed, and (iii) the resulting change in throughput across subflows.

The Linux MPTCPv1 implementation behaves as specified: forged MP_PRIO on one subflow affects only that subflow, attacks require a correct sequence number for each subflow, traffic diversion is still possible by a partial on-path or a full on-path attacker, but only through multiple independent injections, not a single forged message. Thus, the vulnerability is not eliminated but significantly constrained.

5.2.3. Subflow creation via ADD_ADDR

In MPTCPv0 an attacker could insert themselves as a man-in-the-middle by forging ADD_ADDR packets [12]. Since ADD_ADDR carried no authentication, even an off-path attacker could advertise their own IP address to a host. The victim would then initiate a JOIN handshake toward the attacker, creating a subflow under the attacker's control. Once established, the attacker could relay packets and potentially steer or hijack traffic on the connection. To mitigate this, MPTCPv1 added an HMAC field to ADD_ADDR. A forged address advertisement is accepted only if the attacker knows both hosts' MPTCP keys. Therefore, version 1 removes the unauthenticated mechanism for subflow creation, but not the underlying attack goal: an attacker who learns the keys can still attempt to create new subflows.

Using MPTCPcap we reproduced the v0 attack on v1 by computing the correct HMAC using the keys extracted from the MP_CAPABLE handshake. We then injected forged ADD_ADDR packets on a controlled subflow and observed: whether the host accepted the advertised address, whether it subsequently initiated a JOIN toward the attacker, and whether a man-in-the-middle subflow could be formed.

We have seen that MPTCPv1 correctly rejects forged ADD_ADDR packets unless the attacker knows the keys. However, a key-eavesdropping attacker (partial on-path during the MP_CAPABLE handshake) can still exploit the attack exactly as in v0. The vulnerability is therefore not removed; it is merely raised to the same level of difficulty as forging valid MP_JOIN messages.

Fake Subflow Creation via MP_JOIN. Our reassessment uncovered a previously undocumented attack vector: an attacker with knowledge of the connection keys can create fake subflows by forging the three-way JOIN handshake. This method does not require the use of ADD_ADDR and eliminates the need for the attacker to discover valid sequence and acknowledgment numbers for the forged packets.

The attacker performs the following steps: (i) sends a forged MP_JOIN SYN (token derived from the keys), (ii) receives the server's SYN/ACK, (iii) replies with the correct MP_JOIN ACK containing the expected HMAC, (iv) completes a subflow where the attacker impersonates the client.

From the server's perspective, a subflow appears identical to a legitimate one. JOIN authentication relies exclusively on keys and nonces, without considering the source address or routing path; as a consequence, version 1 does not prevent this vulnerability, which can lead to a subflow injection attack.

To verify the above vulnerability, we extended MPTCPcap to craft a full forged JOIN handshake. As expected, the tool was able to extract both keys from the MP_CAPABLE ACK, generate the correct token and HMAC for the JOIN sequence, successfully establish a subflow with the server, and then use that subflow to inject control traffic (e.g., FASTCLOSE) or relay packets.

5.3. New Discovered Vulnerabilities With MPTCPcap

Our tool played a crucial role in identifying and analyzing the following new vulnerabilities:

- **Arbitrary Subflows Removal via REMOVE_ADDR**: An attacker can trigger the shutdown of any subflow with a forged REMOVE_ADDR if the protocol implementation fails to ensure that the subflow in question is no longer active.
- **Subflow Creation Prevention via Full MP_JOIN Handshakes**: Instead of just sending MP_JOIN SYNs, an attacker who has eavesdropped the keys can fully establish fake subflows. While the first method aims to saturate the limit of half-open subflows (typically the number of possible TCP connections), the second aims to saturate the smaller limit of additional fully established subflows (i.e., at most 8 in the Linux implementation).

These tests were conducted on the Linux kernel implementation of MPTCPv1, specifically on Arch Linux with kernel version 6.12.0, the latest available at the time of the start of testing. Additionally, the Apple implementation of MPTCPv1 in the XNU kernel was tested. However, since it currently supports only the client side, not accepting incoming connections, we were unable to test subflow creation prevention attacks.

5.3.1. Arbitrary Subflows Removal via REMOVE_ADDR

The MPTCP REMOVE_ADDR option is intended to signal that a host's endpoint is no longer reachable, prompting the peer to close all subflows bound to that address. To avoid abuse, RFC 8684 [24] recommends (but does not mandate) that implementations first verify reachability using a TCP keep-alive before tearing subflows down. If the keep-alive prompts a response, the receiver should not close the subflows.

Since both Linux kernel and the XNU kernel do not follow these recommendations, two attacks are possible: (i) an on-path attacker could craft a packet containing this option in order to close all subflows, except the one they control, redirecting traffic there (a downgrade attack); (ii) an off-path attacker can similarly close all subflows, or close only some subflows to redirect traffic to others, filling their bandwidth, causing a DoS attack.

Attack Execution on Linux. The Linux MPTCP implementation does not perform the recommended reachability check. Upon receiving a REMOVE_ADDR, Linux immediately initiates subflow teardown, regardless of whether the affected subflows are active and responsive. To verify this, we used MPTCPcap to inject a forged ACK containing a REMOVE_ADDR subtype into active multipath connections. The packet spoofed the peer's 4-tuple and contained the targeted Address ID (trivial to identify, since Linux assigns Address IDs incrementally); sequence numbers were obtained either by interception or by guessing within the receive window.

Linux immediately closed every subflow associated with the advertised address, without performing any reachability check. Figure 5 shows the throughput evolution in a three-subflow experiment: the moment a REMOVE_ADDR targeted at subflow 3 is injected, its throughput drops to zero; the same happens for subflow 2 after a second forged message. In both cases the subflows were actively exchanging data and would have responded to a keep-alive, yet Linux instantly initiated a FIN exchange and removed them.

This behaviour confirms that the Linux implementation does not enforce the RFC-recommended address-reachability test, making the attack successful.

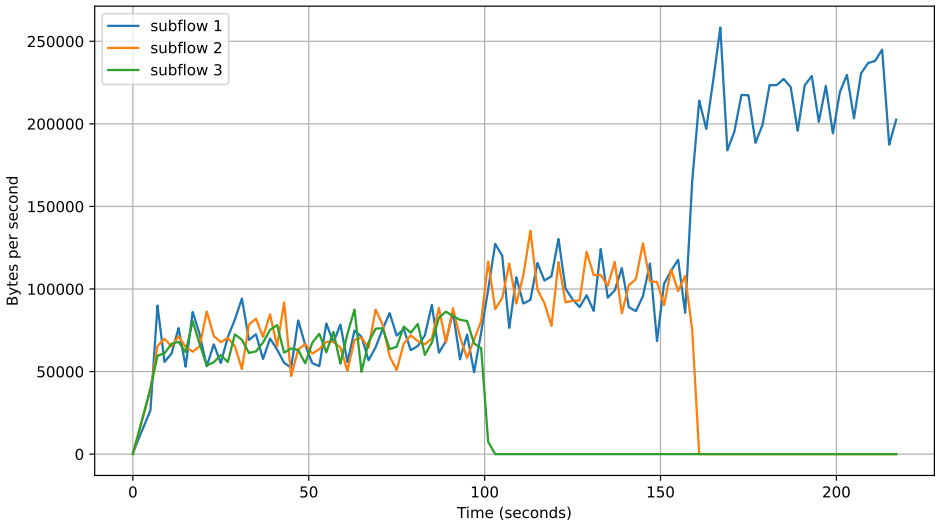

Figure 5: Throughput of subflows during a REMOVE_ADDR attack.

Attack Execution on XNU. The XNU implementation, on the other hand, appears to completely ignore any Address Removal packet. To verify this behavior, we set up a connection where the macOS client established two subflows with the server, each corresponding to one of the server's interfaces. Even the disconnection of one of the server's endpoints and the subsequent sending of a legitimate REMOVE_ADDR triggered no response from the client. This observation is further supported by an analysis of XNU's kernel source code[3], which does not appear to include any handling of the REMOVE_ADDR subtype. Although XNU is locally immune, it remains indirectly vulnerable when communicating with Linux servers: a forged REMOVE_ADDR sent to the Linux endpoint still forces the Linux host to close the subflow, degrading the connection and eliminating multipath benefits.

5.3.2. Subflow Creation Prevention via Full MP_JOIN Handshakes

A multipath connection remains resilient only as long as hosts can create additional subflows. An attacker who already controls or observes one subflow can aim to obtain full control of the entire connection by preventing the hosts from adding any new paths. MPTCPv1 specifies a limit on the number of additional subflows for each connection (e.g., 0–8 on Linux). If an attacker can force the endpoint to exhaust these slots, for example by creating multiple fake subflows, the connection becomes permanently single-path, and therefore easy to monitor or manipulate.

The key observation is that MPTCPv1 does not authenticate the source address of MP_JOIN packets. Any endpoint that can compute the correct token and HMAC (i.e., any attacker who has eavesdropped the initial MP_CAPABLE handshake) may initiate new subflows on behalf of the legitimate peer. This allows an attacker positioned on the first subflow to: (i) Steal the keys in plaintext by observing the MP_CAPABLE ACK; (ii) Forge multiple complete MP_JOIN handshakes with the server, each using a fake address ID and a spoofed source port; (iii) Consume all subflow slots, leaving the server unable to accept any legitimate future Join attempts from the client. Unlike the v0 Join-flooding attack (which relied on leaving half-open subflows), this variant completes each handshake. As a result, the server treats these subflows as valid, stable, and usable.

[3]https://github.com/apple-oss-distributions/xnu

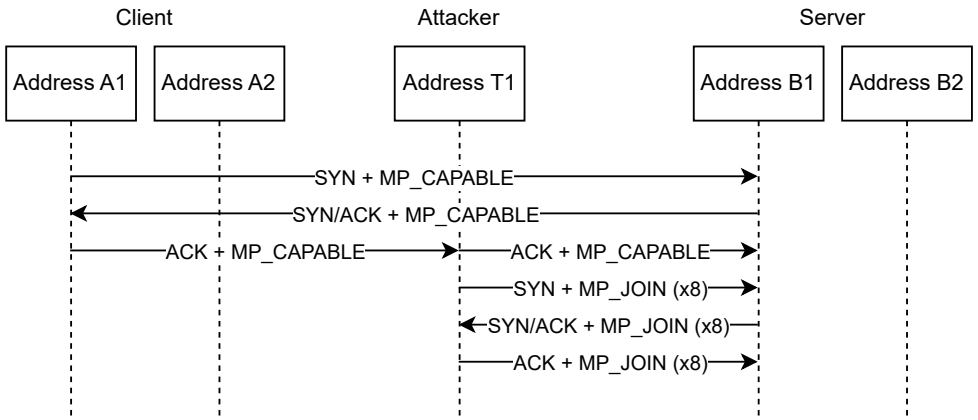

Figure 6: Packets sent in subflow creation prevention attack.

Attack Execution on Linux. This attack requires a partial on-path attacker with temporary visibility of the first subflow during the `MP_CAPABLE` handshake, since the keys must be observed in transit. These are the steps to carry out the attack, also summarized in Figure 6:

1. While on-path on the initial subflow, the attacker intercepts the ACK of the `MP_CAPABLE` handshake, and extracts both hosts' key in plaintext;
2. Immediately after the `MP_CAPABLE` ACK (and before it reaches the client), the attacker sends 8 `MP_JOIN` SYN packets to the server. Each SYN uses the attacker's IP address and a random port, includes a fresh random number, sets the Backup flag and includes a random Address ID. From the server's point of view, these appear to be legitimate Join attempts from the client.
3. The server responds to each accepted `MP_JOIN` with a SYN/ACK containing its own random challenge value;
4. The attacker completes the handshakes. For every SYN/ACK received, the attacker matches it to the corresponding SYN (via the port number), computes the correct HMAC from the concatenation of the client's and server's keys and sends the final `MP_JOIN` ACK. The server now treats each subflow as fully established;
5. Once the allowed number of subflows is reached, the server refuses all subsequent Join attempts from the real client. The attacker has effectively frozen the connection into a single controlled path, preventing any new subflows from being added.

Because the attacker created all subflows as backup, the server did not transmit data on them and never detected that these connections were fake. In effect, the attacker forced the multipath session to remain permanently single-path, pinned to the subflow they already controlled.

Creation of additional attacker-controlled subflows. Once the attacker has saturated the server's subflow table, she can also inject new subflows under her control. To do so, she frees one slot by removing a previously forged backup subflow (via a forged `REMOVE_ADDR`, which Linux accepts without verification). She then advertises her own address to the client using a forged `ADD_ADDR`. The client initiates an `MP_JOIN` toward that address, and the attacker simply forwards each step of the handshake between the client and the server. Both hosts, therefore, complete a valid Join Connection handshake, but the resulting subflow terminates at the attacker. This allows the attacker not only to block legitimate subflows but also to selectively replace them with attacker-controlled ones.

Attack Feasibility on XNU. XNU supports the client side of MPTCPv1 but does not accept inbound `MP_JOIN` handshakes. As a result, the attack cannot be performed against an XNU client.

6. Conclusions

In this paper, we introduced MPTCPcap, a real-time analysis tool specifically designed to assess the security of MPTCP connections. Unlike existing tools that primarily target performance metrics, our tool focuses on security evaluation through the simulation of attacks and the injection of crafted or manipulated packets. Using our tool, we reassessed known attacks and identified two previously undisclosed vulnerabilities. The first, caused by a flaw in the Linux MPTCP implementation, allows an off-path attacker to terminate subflows using Address Removal messages. The second, inherent to the protocol specification, enables an attacker possessing the connection keys to block the creation of additional subflows, effectively limiting MPTCP's multipath capabilities.

While our evaluation focuses primarily on the Linux MPTCPv1 implementation, the framework can be extended to other stacks. A systematic evaluation, including broader testing and additional kernel versions, represents an important direction for future work. Moreover, although we demonstrate the feasibility of some attacks, a more quantitative study of their success probability and timing under different network conditions should be conducted. Future work also includes translating identified vulnerabilities into concrete, implementable mitigation guidelines and refining the user interface to provide more intuitive options for traffic analysis and attack simulation inputs.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT, Grammarly in order to: Grammar and spelling check, Paraphrase and reword. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] A. Ford, C. Raiciu, M. J. Handley, O. Bonaventure, OLD TCP Extensions for Multipath Operation with Multiple Addresses, Request for Comments RFC 6824, Internet Engineering Task Force, 2013. doi:10.17487/RFC6824.

[2] T. Shreedhar, D. Zeynali, O. Gasser, N. Mohan, J. Ott, A longitudinal view at the adoption of multipath TCP, Computing Research Repository (CoRR), arXiv abs/2205.12138 (2022). URL: https://arxiv.org/pdf/2205.12138v1.

[3] F. Aschenbrenner, T. Shreedhar, O. Gasser, N. Mohan, J. Ott, From single lane to highways: Analyzing the adoption of multipath TCP in the Internet, in: IFIP Networking Conference, 2021, pp. 1–9. doi:10.23919/IFIPNetworking52078.2021.9472785.

[4] R. K. Chaturvedi, S. Chand, Multipath tcp security over different attacks, Transactions on Emerging Telecommunications Technologies 31 (2020) e4081. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4081. doi:https://doi.org/10.1002/ett.4081. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/ett.4081.

[5] O. Bonaventure, MPTLS : Making TLS and Multipath TCP stronger together, Internet-Draft draft-bonaventure-mptcp-tls-00, Internet Engineering Task Force, 2014. URL: https://datatracker.ietf.org/doc/draft-bonaventure-mptcp-tls/00/, work in Progress.

[6] M. Jadin, G. Tihon, O. Pereira, O. Bonaventure, Securing multipath tcp: Design & implementation, in: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, 2017, pp. 1–9. doi:10.1109/INFOCOM.2017.8057011.

[7] B. Hesmans, O. Bonaventure, Tracing multipath TCP connections, in: Proceedings of the 2014 ACM conference on SIGCOMM, ACM, 2014, pp. 361–362. doi:10.1145/2619239.2631453.

[8] J. Ishac, mptcplot, 2017. URL: https://github.com/nasa/multipath-tcp-tools/, online; accessed 2025.

[9] M. Coudron, Passive analysis for multipath TCP, in: Proceedings of the Asian Internet Engineering Conference on - AINTEC '19, ACM Press, 2019, pp. 25–32. doi:10.1145/3340422.3343638.

[10] N. Cardwell, Y. Cheng, L. Brakmo, M. Mathis, B. Raghavan, N. Dukkipati, H. keng Jerry Chu, A. Terzis, T. Herbert, packetdrill: Scriptable network stack testing, from sockets to packets, in: 2013 USENIX Annual Technical Conference (USENIX ATC 13), USENIX Association, San Jose, CA, 2013, pp. 213–218. URL: https://www.usenix.org/conference/atc13/technical-sessions/presentation/cardwell.

[11] M. Bagnulo, C. Paasch, F. Gont, O. Bonaventure, C. Raiciu, Analysis of Residual Threats and Possible Fixes for Multipath TCP (MPTCP), Request for Comments RFC 7430, Internet Engineering Task Force, 2015. doi:10.17487/RFC7430.

[12] F. Demaria, Security Evaluation of Multipath TCP, Master's thesis, KTH Royal Institute Of Technology, 2016. URL: https://www.diva-portal.org/smash/get/diva2:934158/FULLTEXT01.pdf.

[13] A. Munir, Z. Qian, Z. Shafiq, A. Liu, F. Le, Multipath TCP traffic diversion attacks and countermeasures, in: 2017 IEEE 25th International Conference on Network Protocols (ICNP), IEEE, Toronto, ON, 2017. doi:10.1109/ICNP.2017.8117547.

[14] M. Z. Shafiq, F. Le, M. Srivatsa, A. X. Liu, Cross-path inference attacks on multipath TCP, in: Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks, HotNets-XII, Association for Computing Machinery, 2013, pp. 1–7. doi:10.1145/2535771.2535782.

[15] Y. Cao, J. Chen, Q. Liu, G. Lei, H. Wang, I. You, Can multipath tcp be robust to cyber attacks with incomplete information?, IEEE Access 8 (2020) 165872–165883. doi:10.1109/ACCESS.2020.3021475.

[16] V. A. Kumar, D. Das, Data sequence signal manipulation in multipath TCP (MPTCP): The vulnerability, attack and its detection, Computers & Security 103 (2021). doi:10.1016/j.cose.2021.102180.

[17] F. Shafique, S. Fatima, F. Y. Khuhawar, Z. A. Arain, An Analysis of Multipath TCP Security Vulnerabilities: A Research Study, in: 2022 IEEE 19th International Conference on Smart Communities: Improving Quality of Life Using ICT, IoT and AI (HONET), 2022. doi:10.1109/HONET56683.2022.10019021, iSSN: 1949-4106.

[18] J. Pedro Meira, R. P. C. Monteiro, J. M. C. Silva, Securing MPTCP Connections: A Solution for Distributed NIDS Environments, in: 2022 IEEE 47th Conference on Local Computer Networks (LCN), 2022. doi:10.1109/LCN53696.2022.9843271, iSSN: 0742-1303.

[19] P. Tomar, G. Kumar, L. P. Verma, V. K. Sharma, D. Kanellopoulos, S. S. Rawat, Y. Alotaibi, Cmt-sctp and mptcp multipath transport protocols: A comprehensive review, Electronics 11 (2022). URL: https://www.mdpi.com/2079-9292/11/15/2384. doi:10.3390/electronics11152384.

[20] P. Abeni, mptcp-tools, https://github.com/pabeni/mptcp-tools, 2019. Online; accessed 2025.

[21] Debian, mptcpize, https://manpages.debian.org/testing/mptcpize/mptcpize.8.en.html, 2021.

[22] Linux, ip-mptcp, http://man7.org/linux/man-pages/man8/ip-mptcp.8.html, 2020.

[23] M. Baerts, mptcpd: Multipath TCP Daemon, https://github.com/multipath-tcp/mptcpd, 2019. Online; accessed 2025.

[24] A. Ford, C. Raiciu, M. J. Handley, O. Bonaventure, C. Paasch, TCP Extensions for Multipath Operation with Multiple Addresses, Request for Comments RFC 8684, Internet Engineering Task Force, 2020. doi:10.17487/RFC8684.