Multipath TCP Traffic Diversion Attacks and Countermeasures

Ali Munir
Computer Science and Engineering
Michigan State University
munirali@msu.edu

Zhiyun Qian
Computer Science and Engineering
U.C. Riverside
zhiyunq@cs.ucr.edu

Zubair Shafiq
Dept. of Computer Science
The University of IOWA
zubair-shafiq@uiowa.edu

Alex Liu
Computer Science and Engineering
Michigan State University
alexliu@cse.msu.edu

Franck Le
IBM T. J. Watson
IBM Research
fle@us.ibm.com

***Abstract*—Multipath TCP (MPTCP) is an IETF standardized suite of TCP extensions that allow two endpoints to simultaneously use multiple paths between them. In this paper, we report vulnerabilities in MPTCP that arise because of cross-path interactions between MPTCP subflows. First, an attacker eavesdropping one MPTCP subflow can infer throughput of other subflows. Second, an attacker can inject forged MPTCP packets to change priorities of any MPTCP subflow. We present two attacks to exploit these vulnerabilities. In the connection hijack attack, an attacker takes full control of the MPTCP connection by suspending the subflows he has no access to. In the traffic diversion attack, an attacker diverts traffic from one path to other paths. Proposed vulnerabilities fixes, changes to MPTCP specification, provide the guarantees that MPTCP is at least as secure as TCP and the original MPTCP. We validate attacks and prevention mechanism, using MPTCP Linux implementation (v0.91), on a real-network testbed.**

I. Introduction

Multipath TCP (MPTCP) is an IETF standardized suite of TCP extensions that allow an MPTCP connection between two hosts to simultaneously use multiple available paths [1]. This parallelism capability significantly improves throughput and reliability while maintaining the same TCP interface to the applications. For example, smartphones can use MPTCP to stream videos over both WiFi and 3G/4G connections simultaneously for better user experience. Since its inception, MPTCP has been fueled with great interest and excitement from both academia [2]–[5] and industry [1], [6]. For example, Apple iOS (version 7 onwards) supports MPTCP and Google Android now has modified ROMs with MPTCP [7], [8] support.

Compared to TCP, MPTCP splits the security risk through multiple subflows. If only one subflow is attacked, the MPTCP connection as a whole may not be severely affected. In theory, if only one subflow is eavesdropped, the attacker should not be able to eavesdrop the traffic of the whole MPTCP connection. On the flip side, MPTCP has a larger attack surface than TCP – as long as *any one* of the subflows is under attack (*e.g.* by a man-in-the-middle attacker), the whole MPTCP connection can be affected, which puts MPTCP at a disadvantage compared to TCP. Prior work lacks on analyzing and comparing the attack surface of MPTCP with that of TCP [9], [10]. To date, it is still largely unclear whether new attacks are possible or whether prior known attacks on TCP (*e.g.* connection hijack [11]–[14]) can be more easily launched on MPTCP [1].

To fill this gap, in this paper, we specifically analyze the MPTCP's *cross-path attack surface*, *i.e.* what an attacker controlling one path can do to affect other paths or the MPTCP connection as a whole. We show that there are indeed worrisome cross-path attacks that make the current version of MPTCP less secure than TCP. For instance, we show that an attacker controlling *any one* of the subflows can take full control over the MPTCP connection (See § IV-B), which makes many attacks that require full man-in-the-middle capability much more likely to occur in MPTCP [15]–[17]. We subsequently analyze the root causes of such cross-path vulnerabilities, propose and implement a set of changes to MPTCP to prevent such class of attacks. As the weaknesses are fundamental to MPTCP, we argue that these changes should be made at the MPTCP layer itself (instead of being pushed to higher layers).

Our key contributions and findings are as follows.

• *Vulnerabilities:* We report two cross-path MPTCP vulnerabilities. First, an attacker eavesdropping one MPTCP subflow can infer the throughput of other MPTCP subflows by analyzing the local and global sequence numbers (GSN) embedded in MPTCP headers. Second, an attacker can send forged packets with the MPTCP `MP_PRIO` option containing the backup flag to set any MPTCP subflow as the backup, causing it to be stalled completely. Even though the vulnerabilities are verified on the Linux MPTCP implementation (version 0.91), they are actually caused by the current design of MPTCP, which calls for changes to the specification rather than simple implementation fixes.

• *Attacks:* Based on these vulnerabilities, we present two practical attacks on MPTCP, directed traffic diversion attacks and hijack attacks. We show that an attacker can onload or offload controlled amount of traffic on specific MPTCP subflows. We also show that an attacker controlling one path can easily force all traffic to route through itself by setting all other subflows as backup. We implement and validate these attacks on a real network testbed using MPTCP Linux Kernel implementation (v0.91) [18].

• *Attack Prevention and Countermeasures:* We show that the two cross-path vulnerabilities can in fact be prevented by leveraging the "secret splitting" across multiple paths. The intuition is that it is highly unlikely that an attacker will be able to monitor and control all paths used by an MPTCP connection. Therefore, secrets can be split across multiple paths to prevent complete eavesdropping or tampering. We propose a set of changes to MPTCP based on the intuition and implement in the Linux MPTCP code base to verify the ability to defend against this class of cross-path attacks.

In the rest of the paper, we present MPTCP background and related work in § II. We present our threat model in § III, and present MPTCP cross-path attacks in § IV. Lastly, we discuss attack countermeasures in § V and conclude in § VI.

II. MPTCP Background & Related Work

Below, we first provide a brief overview of MPTCP, and then discuss security issues of MPTCP in prior literature.

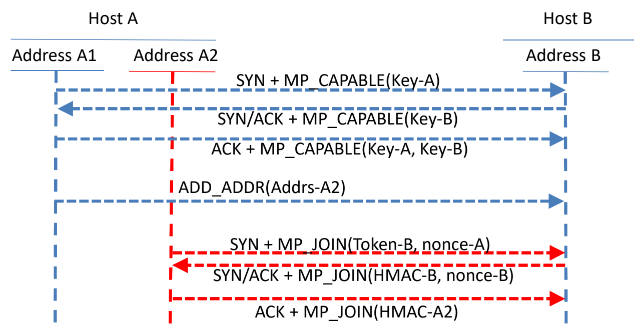

Host A
Address A1
Address A2
Host B
Address B
SYN + MP_CAPABLE(Key-A)
SYN/ACK + MP_CAPABLE(Key-B)
ACK + MP_CAPABLE(Key-A, Key-B)
ADD_ADDR(Addrs-A2)
SYN + MP_JOIN(Token-B, nonce-A)
SYN/ACK + MP_JOIN(HMAC-B, nonce-B)
ACK + MP_JOIN(HMAC-A2)

Fig. 1. Example 2-path MPTCP connection setup

A. MPTCP Background

MPTCP is an extended version of TCP implemented using the TCP option field, which allows a TCP connection between two hosts to use multiple paths simultaneously [1]. Below we explain the operation of an example 2-path MPTCP connection, illustrated in Figure 1.

Connection Establishment. The client uses its IP address $A1$ to establish an MPTCP subflow with the server. The SYN, SYN/ACK, and ACK handshake packets include the MP_CAPABLE option to indicate MPTCP support. Hosts also exchange 64-bit keys during the handshake to authenticate the establishment of additional subflows in future. The client notifies the server that it has another IP address $A2$ using the ADD_ADDR option. Note that, these address advertisements can be refreshed periodically and sent on any subflow. A host receiving the advertisements will then establish new MPTCP subflows using the MP_JOIN option. The SYN, SYN/ACK, and ACK handshake packets include the MP_JOIN option to indicate establishment of the new MPTCP subflow, using Adress A2. Hosts use 32-bit tokens to identify the MPTCP connection. Hosts also use 64-bit truncated Hash-based Message Authentication Code (HMAC) for authentication. Finally, MPTCP uses address identifiers (advertised via ADD_ADDR), instead of IP addresses, to cope with possible IP address changes due to the presence of a NAT on the path. Each address has a corresponding address identifier to uniquely identify it within a connection.

Data Transfer. MPTCP packets include a 32-bit local sequence number and a 64-bit global sequence number (also called data sequence number, short for DSN) to ensure reliable and in-order data delivery. The 32-bit local sequence number is used for flow control at the subflow level and 64-bit global sequence number is used for flow control at the connection level. MPTCP also signals both the connection-level acknowledgements and subflow level acknowledgments to implement flow control.

Subflow Priority. MPTCP uses the MP_PRIO option to dynamically change the priority of subflows. Hosts can request change in the priority to use a subflow as regular or backup. A backup subflow is used only if there is no regular subflow available for data transmission [1]. MP_PRIO carries an optional address ID field that can be used to specify the subflow that needs to be set as backup. To set a subflow as backup, a host can request a change in the subflow priority by sending MPTCP MP_PRIO option to the other host with the corresponding address identifier. The key property of MP_PRIO messages is that they can be sent on any subflow. The reason is that if a subflow is already congested, it may not be possible to deliver the message to pause itself.

B. MPTCP Attacks

RFC 6181 [9] and RFC 7430 [10] discuss some MPTCP-specific attacks including flooding, hijacking, and Denial of Service (DoS) attacks. However, they do not cover the threat model and the problems discussed in this paper.

Shafiq *et al.* [19] proposed cross-path throughput inference attacks on MPTCP. The authors showed that MPTCP subflows are interdependent due to the congestion control coupling. The authors demonstrated that the resulting side-channels can be exploited by an attacker controlling one of the subflows to infer the throughput of other subflows. In this paper, we build on their work by proposing novel traffic diversion attacks optionally based on cross-path MPTCP throughput inference that provides feedback to the attacker. We also design and implement countermeasures to defend against these cross-path throughput inference based attacks.

Popovici *et al.* [20] discuss use of MPTCP congestion control for diverting traffic from one network to another to gain profit. Authors introduce a policy drop strategy that, a network operator, can be used to divert traffic from one path to another path by introducing packet drop on one of the paths. This allows network operators to reduce the traffic they carry, and divert some of their own traffic to the other path(s). However, this work does not consider per-connection policy drops and assume all the connections experience same level of packet drops. Our MPTCP connection throughput inference attack will allow more fine-grained control of traffic diversion attacks. Network operators can use throughput inference to monitor the overall connection throughput, from the subflow through their network, and only divert some amount of traffic that does not hurt the overall MPTCP connection throughput. Moreover, using MP_PRIO option, an attacker can diverge traffic to or from a specific path, which can not be done by other existing attack scenarios.

Jadin *et al.* [21] propose MPTCPSec that adds authentication and encryption to MPTCP for both the data and MPTCP options. MPTCPSec is a complete suite that can be used to prevent the attacks discussed in this paper. It is a relatively heavyweight solution (e.g., key distribution/sharing and more complete encryption) compared to the small changes we propose to prevent the reported attacks.

Since MPTCP transparently distributes traffic of an end-to-end TCP connection across multiple paths, network monitoring devices such as intrusion detection systems (IDS) can only observe partial traffic. Therefore, MPTCP will negatively impact the functionality of network monitoring devices [22]. For examples, attackers can evade signature-based detection by splitting malicious payloads across multiple paths. Distributed signature-based intrusion detection approaches (e.g., [23]) have been investigated to overcome these difficulties.

III. MPTCP Threat Model and Vulnerabilities

In this section, we first discuss our threat model for attacks on MPTCP connections and then discuss two vulnerabilities that can be exploited under this threat model.

A. Threat Model

Figure 2 illustrates an MPTCP connection where the attacker is on one of the two paths used in the MPTCP connection. In the more general case, if there are m paths (with m subflows), an attacker may be on a subset of the paths. Our threat model assumes that the attacker can eavesdrop or inject traffic on the paths. This threat model is commonly found in the case of open WiFi access points where an attacker connected to the same access point can eavesdrop or inject traffic on the subflow established through the access point. Our threat model does not assume that the attacker can modify existing traffic.

B. Backup Flag Vulnerability

MPTCP supports using subflows as backup – i.e., using a subflow to send data only if there is no other subflow available. Specifically, an MPTCP host can send a request to the other host to set *any* subflow as a backup by using the MPTCP `MP_PRIO` option. As mentioned earlier, this request can be sent via *any* MPTCP subflow. After receiving such a control packet, the sender will stop sending data and the corresponding subflow's throughput will drop to zero. Unfortunately, unlike `MP_JOIN`, such a control packet has no authentication required by the specification whatsoever [1], allowing an attacker controlling *only one path* to set *any* subflow (using the corresponding address ID) as back up, Figure 2. For example, for the two path MPTCP connection shown in Figure 2, the attacker on path1 can send a forged `MP_PRIO` packet with address ID of path2 to set subflow on path2 as backup. The backup flag vulnerability allows an attacker to divert traffic among MPTCP subflows. An attacker can offload traffic from the eavesdropped subflow to other MPTCP subflows by sending forged packets with the `MP_PRIO` option to set the eavesdropped subflow as a backup. An attacker can also onload traffic from non-eavesdropped subflows to the eavesdropped subflow by sending forged packets with the `MP_PRIO` option to set all other subflows as backup. This attack is similar to connection hijack attack, where an attacker diverts all the traffic to pass through the subflow on the eavesdropped path.

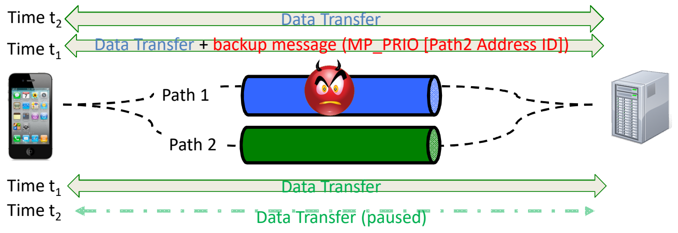


Fig. 2. Subflow pause using MP_PRIO option

C. Throughput Inference Vulnerability

If there are two subflows in an MPTCP connection, Shafiq et al. showed that an attacker eavesdropping a subflow can learn the throughput of the other subflow [19]. For example, an attacker can infer the throughput of the non-eavesdropped subflow on most phones that have both WiFi and cellular network interfaces. The subflows of an MPTCP connection are seemingly independent, but in reality they are fundamentally coupled because they carry data of the same MPTCP connection. Take the example of an MPTCP connection with two subflows, since the two subflows carry packets in the same MPTCP connection, the destination host needs to reassemble the packets from both subflows into that MPTCP connection. Thus, each packet in a subflow needs to have two sequence numbers: a local sequence number for reliable transmission over that subflow, which is the traditional subflow sequence number encoded in the TCP sequence field, and a global sequence number for assembling packets from both subflows into that MPTCP connection. By passively eavesdropping one subflow, an attacker can calculate the throughput of the eavesdropped subflow from the local sequence number and that of the MPTCP connection from the global sequence numbers.

In this paper, we show that it is not hard to generalize the result to more than two subflows. Specifically, if an attacker eavesdropping on $m-1$ subflows for an m-path MPTCP connection ($m \geq 2$), it is possible to infer the throughput of the non-eavesdropped subflows. For an eavesdropped subflow, from the local sequence numbers, which indicates the number of bytes that have been transferred over that subflow, the attacker can calculate the throughput of the subflow; from the global sequence number, which indicates the number of bytes that have been transferred over all subflows, the attacker can calculate the throughput of the overall MPTCP connection. Let $t_1, \cdots, t_m$ denote the throughput of m subflows over paths $p_1, \cdots, p_m$, respectively. Let t denote the throughput of the overall MPTCP connection over m paths. Then we have $t_1 + ... + t_m = t$. Suppose the attacker can eavesdrop the $m-1$ paths $p_1, \cdots, p_{m-1}$. Then, the attacker can calculate the throughput t_m of the subflow over path p_m as $t_m = t - \sum_{i=1}^{m-1} t_i$.

The ability to infer throughput can help an attacker validate the success of connection hijack attack and traffic diversion attacks. Note that, a simple approach to validate the attacks is to check that there are no holes in DSS space, however, it works only for a connection with two subflows or when all the subflows have been hijacked. For a connection with more than two subflows or when only specific subflows are targeted, we still need throughput inference to validate and detect change in throughput.An attacker eavesdropping a single path can infer the total throughput of the MPTCP connection (t) and the throughput of the subflow (t_m) on that path. Therefore, using this information, an attacker can validate that the backup flag attack was successful. For example, for an MPTCP connection with two subflows, t will be equal to t_m if the backup attack was successful. Similarly, from this the attacker can calculate the total throughput of the MPTCP connection on other paths (as $t_{paths} = t - t_m$), however, an attacker can not calculate the throughput of a particular subflow on the uneavesdropped path, unless it can eavesdrop $m - 1$ paths. For example, an attacker eavesdropping $m-2$ paths can not calculate the exact throughput of the other 2 paths as it does not have sufficient information to compute the throughput. This information can help an attacker to offload or onload traffic from one subflow to another subflow while maintaining the same overall throughput of the MPTCP connection. Note that the connection hijack attack can work without throughput information, however throughput information can help an attacker make sure that the attack was successful. For example, in some cases, the host may or may not decide to stop sending the data on the backup path. Therefore, it is important for connection hijack attack that the attacker is able to infer the throughput.

IV. Attack Analysis and Validation

In this section, first, we demonstrate how an attacker can leverage passive cross-path throughput inference and backup flag vulnerability to launch a connection hijack attack. Next, we demonstrate how an attacker can launch directed traffic diversion attacks. In the hijack attack, an attacker diverts all the traffic through his own path to take full control of the connection. In the directed traffic diversion attack, an attacker manipulates traffic on specific individual subflows while maintaining the same throughput of the MPTCP connection. Note that the connection throughput can itself vary in real-life. However, using throughput inference, an attacker can ensure that the overall throughput does not change too much (for long) and any unusual activity is not discovered by the attack victim.

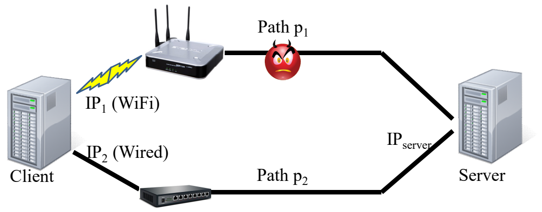


Fig. 3. Experiment Topology

A. Experimental Setup

To validate the hijack and traffic diversion attack under varying network settings, we conduct experiments on a testbed running the MPTCP Linux implementation (v0.91) [18]. We consider the scenario shown in Figure 3, where a client and server can communicate using two paths configured as WiFi (path p_1) and Wired (path p_2). We setup and configure the loss rate and throughput on these paths to approximate WiFi and Wired connections. MPTCP connections compete with single-path TCP connections on paths p_1 and p_2. The client and the server can have a 2-path MPTCP connection whose subflows use paths p_1 and p_2. Let t_1 and t_2 denote the throughput of the MPTCP subflows over the paths p_1 and p_2 respectively. Let T_1 and T_2 denote the throughput of TCP connections passing through paths p_1 and p_2 respectively. In our experiments, we assume that there is a malicious attacker on path p_1 who can eavesdrop and inject forged packets on path p_1.

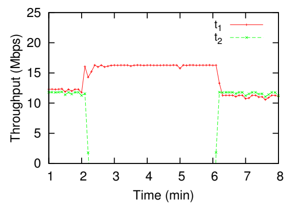


Fig. 4. MPTCP connection hijack attack using backup flag

B. Connection Hijack Attacks

First, we show a special case of traffic diversion, where an attacker can launch a connection hijack attack to onload traffic to the subflow on its eavesdropped path.

1) Motivation: In these scenarios, an attacker's motivation could be to observe all the data in the MPTCP connection for surveillance or to charge users for additional data usage as a cellular ISP. To gain full control over MPTCP connection, an attacker can set the subflows as backup and pause non-eavesdropped subflows between two hosts. By pausing the non-eavesdropped subflow, an attacker can get full control over the MPTCP connection.

2) Overview: In the connection hijack attack, the attacker uses the backup flag vulnerability to onload traffic to its own path and uses the cross-path throughput inference to validate the success of hijack attack.

Consider the scenario shown in Figure 3 where two subflows of an MPTCP connection pass through paths p_1 and p_2. An attacker can hijack the non-eavesdropped subflow on p_2 by dynamically changing the priority of a subflow and declaring it as backup. To launch this attack, an attacker only needs to know the address identifier of the host, which can be obtained by eavesdropping other paths or easily guessed as they are set incrementally in current Linux implementation of MPTCP. Since the identifier has only 8 bits, it can even be bruteforced. To set a non-eavesdropped subflow as backup between hosts A and B, an attacker can request a change in the subflow priority by sending MPTCP `MP_PRIO` option to host B with the address identifier of host A. Since by design the `MP_PRIO` messages can be sent on any subflow, the attacker eavesdropping on any path capable of sending such a forged backup message can pause any other path. Note that a backup MPTCP subflow may still be used for data transmission later, which can be detected by the attacker by observing the overall MPTCP throughput.

Effectively, this attack degrades an MPTCP connection to a regular TCP connection as all traffic will be routed through the attacker-controlled path. Given that MPTCP has an increased attack surface, this vulnerability makes MPTCP more likely to be hijacked compared to TCP. Unfortunately, MPTCP specification does not include any authentication mechanism whatsoever regarding the `MP_PRIO` message.

3) Attack Validation: To validate the attack, we consider a scenario where the client and the server have a 2-path MPTCP connection whose two subflows use paths p_1 and p_2. Figure 4 plots the results of the hijack attack using backup flag for the scenario shown in Figure 3. In Figure 4, both t_1 and t_2 achieve similar throughput till $t = 2$ minutes. To gain full control, at $t = 2$ minutes, the attacker launches backup flag attack and as a result p_2 throughput drops to zero, which can be observed by the attacker on p_1 using our throughput inference scheme.

The attacker can unset the backup flag to resume the subflow on p_2 ($t = 6$ minutes).

C. Directed Traffic Diversion Attacks

Next, we show how an attacker can launch a *directed* traffic diversion attack on a specific subflow to offload or onload traffic from one subflow to another subflow. As compared to simple traffic diversion, the goal of the directed traffic diversion attack is to hurt the performance of a subflow on a specific path or a subflow used by a specific network. Our threat model is different from (also more generic) [20] and can be used to launch both the simple and directed traffic diversion attacks.

1) Motivation: There are both malicious and benign motivations for traffic diversion.

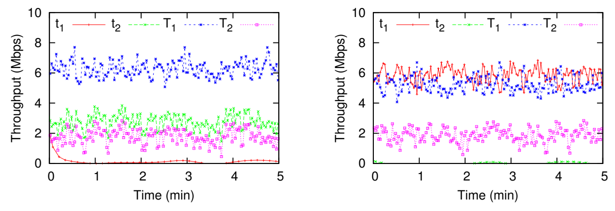
A malicious attacker eavesdropping an MPTCP subflow can launch the directed traffic diversion (offloading) attack to gain more bandwidth, *i.e.*, diverting traffic to other paths so that he can use that path to carry more of his own traffic and at the same time affect the performance of connections using a specific network to which attacker has no access. This will improve the network performance of attacker, however, it will hurt the performance of the users in the targeted network to whom traffic is diverted. Likewise, a malicious ISP can launch the traffic diversion attack to reduce its load by diverting traffic to other ISPs. This allows a malicious ISP to carry more traffic of its own users and degrade the performance of users in the other network.

In a benign scenario, a cellular ISP can use traffic diversion to reduce users' data usage over 3G/4G links by diverting traffic to WiFi links. This helps the users keep their cellular data usage under control and avoid overage charges, and also helps the cellular ISP to offload traffic during peak hours.

In traffic diversion attacks, the attackers can make sure that end-user quality of experience is not affected. This can be made possible by diverting only a limited amount of traffic such that the overall throughput of the MPTCP connection remains unchanged. An attacker can monitor the overall connection throughput from global sequence numbers using our proposed throughput inference model.

2) Overview: To launch the directed offloading or onloading attack, an attacker needs to identify the target path to offload or onload data. For directed offloading data to a specific path, an attacker can either throttle traffic of the paths (e.g., introducing packet loss), he has access to, or set the subflows going through these paths as back up using `MP_PRIO` option in MPTCP. For onloading traffic, an attacker can set targeted paths as backup to direct traffic to its own paths.

To decide which path to onload from or offload to, an attacker can gather mapping between IP address and address ID from the `MP_JOIN` and `ADD_ADDR` options. Both the options contain the IP address and corresponding address ID and are sent over the existing connection in plain-text. Therefore, an attacker eavesdropping on one of the subflows may be able to see this information, if advertised on the eavesdropped path, and find mapping between IP address and its corresponding address ID to launch the targeted traffic diversion attacks.



(a) Offloading (b) Onloading

Fig. 5. Demonstration of the traffic diversion attack

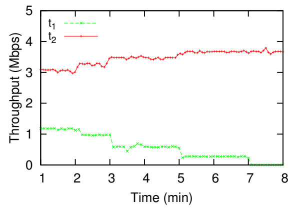
3) Attack Validation: We first consider a scenario where the client and the server have a 2-path MPTCP connection whose two subflows use paths p_1 and p_2. Let t denote the throughput of the MPTCP connection where $t = t_1 + t_2$. We also setup TCP connections (acting as background traffic) passing through paths p_1 and p_2, with throughput T_1 and T_2 respectively. The attacker can eavesdrop path p_1 but not p_2. By eavesdropping, the attacker can calculate the overall throughput t of the MPTCP connection. Using our MPTCP throughput inference scheme, the attacker can then calculate $t_2 = t - t_1$. The attacker can use this information for traffic diversion to make sure that the backup flag vulnerability is exploited correctly and the attack was successful.

Figure 5 plots the results of the traffic diversion attack for two-path connection scenario. To launch the traffic diversion attack, MPTCP subflow passing through p_1 or p_2 is throttled using `MP_PRIO` option. In Figure 5(a), t_1 and t_2 achieve a throughput of about 6 Mbps and 2 Mbps, respectively, and the spare capacity is utilized by TCP flows on both the paths.

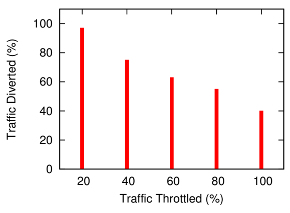
Later, when the attacker sets its own path as backup (*e.g.* to free up more bandwidth on its own path), the MPTCP subflow on path p_1, to offload its traffic to other path, t_1 decreases significantly; however, t_2 increases to make up for the reduced throughput of t_1. However, in this case as the available bandwidth is not sufficient, not all the traffic is offloaded to p_2, the overall throughput, $t = t_1 + t_2$ drops during the traffic diversion attack. Attacker can detect this change and make sure the attack was successful. Similarly, Figure 5(b) demonstrates that when the attacker sets other path as backup, the MPTCP subflow on path p_2, to onload traffic from other path, t_2 drops to zero and t_1 increases to make up for the reduced throughput of t_2. In this case as the available bandwidth is sufficient on p_1, therefore, more than 80% traffic is offlaoded to p_1, and the overall throughput, t remains almost unchanged during the traffic diversion attack. Attacker can detect the drop in t_2 using our throughput inference model and make sure the attack was successful.

4) Step-wise traffic divergence: Step-wise divergence can be used to offload data in steps without realizing any change in throughput. An attacker can launch this attack by gradually throttling the throughput of the desired subflow. By gradual throttling, a two-path MPTCP connection, the attacker can identify the minimum value for t_1 that does not impact t. There are multiple ways to throttle a subflow, such as the attacker can rate limit a subflow by dropping packets, which will cause throughput degradation due to the congestion control mechanisms, which we use in our experiments.

Our experiment demonstrates that an attacker can offload data to other paths without hurting t using the proposed traffic diversion method. Note that the magnitude of possible



(a) With Stepwise Diversion



(b) Throughput Diverted (%)

Fig. 6. Demonstration of stepwise traffic diversion attack.

traffic diversion depends on the properties of path p_2, *i.e.*, an attacker might not be able to offload enough data during higher congestion. To illustrate this, we repeat the above experiment but throttle t_1 in a stepwise manner. We throttle the subflow by dropping the packets of MPTCP subflow and increase the drop rate stepwise to achieve different level of throttling. Similar effect can be achieved by sending duplicate Acks on the path, and as a result the subflow throughput will be dropped. From Figure 6, we can observe that initially all of the throttled traffic is diverted to the other path (i.e., t_2 in this case); however as the attacker increases throttling level, the fraction of the traffic diverted to other path decreases (Figure 6(b)). In Figure 6(a), when the attacker throttles the MPTCP subflow on path p_1 by 20% at time $= 2$ minutes, t_1 decreases and t_2 increases to make up for the reduced throughput of t_1. However, when the attacker increases the throttling level at time $= 3, 5$ and 7 minutes respectively, not all the traffic is diverted to the path p_1. This experiment shows that the amount of diverted traffic may vary depending on path conditions and that all of the traffic may not be diverted if enough capacity is not available.

V. Countermeasures

The problems of throughput inference and connection hijack are rooted in the MPTCP specification. In this section, we propose countermeasures to address these vulnerabilities and as a result prevent the traffic diversion and connection hijack attacks. At a high level, our security improvement hinges on the simple fact that a secret can be split and distributed across multiple paths so that an attacker controlling only a subset of the paths will not be able to know the assembled secret. We show how this idea can be applied to authenticating the `MP_PRIO` control packets as well as encrypting the global sequence number (GSN).

A. Current MPTCP Security Mechanism

MPTCP does have a built-in authentication mechanism that prevents unauthorized subflows to be established. In the beginning of a new MPTCP connection (*i.e.* the very first subflow), the sender and receiver exchange a set of keys (*i.e.* sender key and receiver key) in plaintext at the connection setup time through `MP_CAPABLE` messages during the TCP handshake process. Additional subflows start in the same way as initiating a normal TCP connection, but the SYN, SYN/ACK, and ACK packets also carry the `MP_JOIN` option as well as the authentication information. Specifically, for a new subflow, the `MP_JOIN` on the SYN packet contains a token, a random number, and an address ID. The token is generated from a HASH function of the receiver's key and is used to identify the MPTCP connection to which the new subflow belongs. The random number (nonce) is sent as a challenge to the receiver who needs to use it to compute a subsequent HMAC which prevents replay attacks. Upon receiving an `MP_JOIN SYN` with a valid token, the host responds with a `MP_JOIN` SYN/ACK containing a random number and a truncated (leftmost 64 bits) Hash-based Message Authentication Code (HMAC) based on the sender's and receiver's key as well as the sender's random number. If the token or HMAC is incorrect, the initial sender can reset the subflow and fall back to TCP or it can deny the request for new subflow setup.

As we can see, the only "secrets" (keys) are exchanged during the very first subflow; subsequent subflows only authenticate themselves but no additional new secrets are being shared. This means that as long as an attacker controls the first subflow, he or she can manipulate all subflows (*e.g.* creating fake subflows, or sending other control packets such as `ADD_ADDR` that require authentication). As will be discussed next, our proposed solution requires new secrets to shared on newly established and authenticated subflows. Further, the changes mostly piggyback on the existing security mechanisms without substantial redesign of the protocol.

B. Backup Flag Vulnerability Prevention

As mentioned before, the fundamental reason for backup flag attack is the lack of authentication of `MP_PRIO` messages. We present progressively-more-secure changes to the way `MP_PRIO` messages are generated and handled.

1. Address ID removal. `MP_PRIO` messages use address identifiers (ID) to specify a target subflow for which priority needs to be changed. A simple solution to prevent this attack is to simply remove the address ID from the `MP_PRIO` option, i.e. the `MP_PRIO` option must be sent on the same subflow as the one where it applies. This implies that the attacker must be on-path of this subflow to cause it to become a backup (which he can do anyways). This defense is simple to implement, however, as suggested in RFC6824 [1], address ID is optional for the scenario where a path is extremley congested (e.g., radio coverage issue) that the `MP_PRIO` message has to be delivered on a different path. This proposal has been adopted by RFC6824bis [24].

2. Invalid Address ID rejection. `MP_PRIO` messages use address identifiers (ID) to specify a target subflow for which priority needs to be changed. One possibility to counter an attacker that sends random address IDs could be, for an MPTCP implementation, to reject a subflow where an `MP_PRIO` refers to an invalid address id. Rejecting/terminating a subflow where an `MP_PRIO` with one or more invalid address ID is observed can be a simple solution, as it indicated a compromised path. However, it is unclear whether this can lead to false alarms (especially when significant dynamics are present, e.g., new subflows joining and old ones leaving).

3. Address ID randomization. Currently, address IDs are assigned incrementally (not required by RFC6824 [1]) in MPTCP Linux implementation, which makes it easier for the attacker to guess the address ID of the subflows on the non-eavesdropped path. For example, if the first address is assigned an address ID as 1, the next address of the same connection will be assigned an address ID of 2. This behavior is not defined in the MPTCP specification but rather observed in the current Linux MPTCP implementation. We can mitigate this flaw by randomizing the address IDs that are assigned to the address in MPTCP connection. Although this approach will not

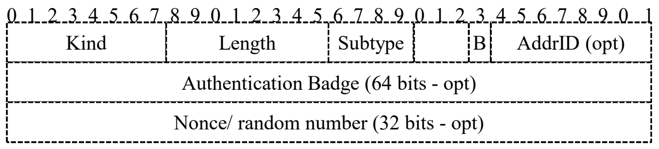


Fig. 7. Secure MP_PRIO TCP option

completely eliminate the threat, it can at least make it harder for the attacker to guess correct address ID of target subflow. We can make it more difficult to guess the correct address ID by adopting techniques such as limiting the number of IDs that can be attempted per fixed time interval. The firewalls and middleboxes can also be configured to avert this attack. However, the current address ID field is only 8 bits and an attacker can still quickly launch this attack.

4. Adding a standard/static badge for authentication. A better solution is to require a subflow level authentication badge, which is sent along with the `MP_PRIO` packet, to ensure that an attacker controlling only one subflow cannot spoof a valid packet. This badge can be similar to the HMAC computed in `MP_JOIN` messages during the subflow setup time. However, there are two issues with using the standard badge, similar to current token. First, the current token is static as it is simply a hash of the received key (and will be the same no matter on which subflow it is used). Unfortunately, the current token is exposed during the subflow setup in either the `MP_JOIN` or `MP_CAPABLE` packet, allowing an attacker controlling any subflow to be able to replay the token. The problem is that the current token is supposed to be used only once during the subflow setup phase, while it is now misused multiple times (as a badge) to authenticate control packets such as `MP_PRIO`. Second, even if the badge is replay-resistant, as long as it is generated from the key exchanged in the first subflow, an attacker knowing the key can always compute the badge easily and set any other subflow as backup.

5. Adding a secure/dynamic badge for authentication. Next, we propose a secure badge based authentication that must meet the following requirements: First, the badge needs to be replay-resistant. Second, the badge generation should not depend on keys shared on only a specific subflow (e.g., primary one).

Choice of badge generation. To satisfy the first requirement, we propose to generate an authentication badge using HMAC derived from not only the keys as well as an additional nonce (32 bits) that changes every packet (e.g., a new field in the `MP_PRIO` message, Figure 7). This avoids sending a fixed badge over the MPTCP connection and prevents replay attacks. We propose using a badge size of 8 bytes, similar to current `MP_JOIN` option in the current MPTCP specification. Similar to the way current token is generated, we can use the truncated HMAC (leftmost 64 bits) using the key material plus nonce, as shown in the example below.

Choice of secret splitting. To satisfy the second requirement, we propose to exchange independent keys on individual subflows and use the combined key to compute the badge. Specifically, since only the primary subflow exchanges keys explicitly through `MP_CAPABLE` messages, we need a way to exchange new keys for non-primary subflows. A simple strategy is to reuse the nonces exchanged in `MP_JOIN` as keys (as they are random in nature too). Once we have a pair of independent keys exchanged on each subflow, the new badge

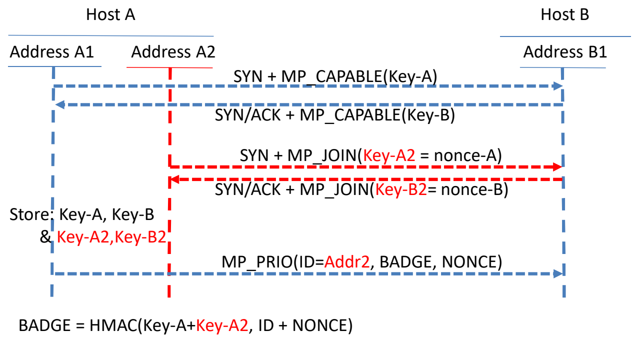

Host A
Host B
Address A1
Address A2
Address B1
SYN + MP_CAPABLE(Key-A)
SYN/ACK + MP_CAPABLE(Key-B)
SYN + MP_JOIN(Key-A2 = nonce-A)
SYN/ACK + MP_JOIN(Key-B2= nonce-B)
Store: Key-A, Key-B
& Key-A2,Key-B2
MP_PRIO(ID=Addr2, BADGE, NONCE)
BADGE = HMAC(Key-A+Key-A2, ID + NONCE)

Fig. 8. Example Use of Secure MP_PRIO

can be computed by combining keys of two subflows: (1) the one on which `MP_PRIO` is sent; (2) the one that is to be set as backup. This means that even if the attacker eavesdrops on the first subflow, it is impossible to compute a valid badge to set any other subflow as backup, so long as the attacker is not eavesdropping on other subflows. For bookkeeping, the additional pair of keys for each subflow needs to be maintained until the subflow is terminated.

Example scenario. Now let us consider an example to understand how everything works together, as shown in Figure 8. Assuming an MPTCP connection between hosts A and B with two subflows 1 and 2, where the attacker controls the subflow 1. First, the keying material is taken from the keys in `MP_CAPABLE` option of subflow 1 and the random numbers from the `MP_JOIN` of subflow 2. To change the priority of subflow 2, the secure `MP_PRIO` will contain an authentication badge generated using keying material and the random nonce, as shown in Fig. 7. The authentication badge is the HMAC derived from the `MP_PRIO` nonce and the keys exchanged on both subflows. For example, in this case, the badge for subflow 2 can be generated as follows: $Badge_2 = HMAC(key_A + key_{A2}, NONCE + ID)$, where key_A and key_{A2} are simply concatenated. If a legitimate `MP_PRIO` message has been sent on subflow 1 (to change the priority of subflow 2), an attacker eavesdropping on 1 may learn the badge. However, in order to inject a subsequent `MP_PRIO` message, the attacker needs to compute a new badge (using the new nonce). Further, since the badge also includes the keys exchanged on subflow 2 during `MP_JOIN`, the attacker cannot easily compute the new badge, unless the attacker is also eavesdropping on subflow 2 (in which case the attacker already has full control of both subflows and the attack is unnecessary).

C. Throughput Inference Vulnerability Prevention

The fundamental enabler for cross-path throughput inference is the GSN embedded in MPTCP packets. On one hand, they are needed for reassembling packets from multiple subflows into a MPTCP connection and for detecting packet losses. On the other hand, as we demonstrated, GSN create a vulnerability that allows an attacker to infer the throughput of a subflow that he has no access to.

Packet header encryption using solutions such as IPsec [25] could prevent attackers from inferring throughput. However, this requires additional keys to be established ahead of time and the VPN server can be a single point of failure. In addition, from the VPN server to the real destination, MPTCP may still be used and therefore encounter same problem. Note that using SSL/TLS will not help as they are above the transport layer and the proposed attacks will still work at the transport layer.

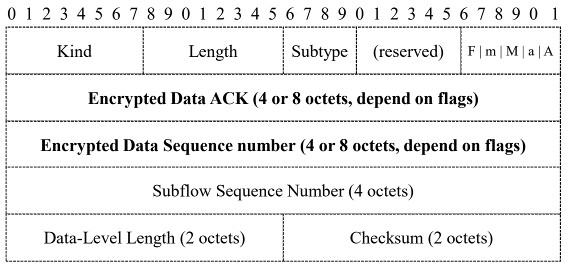

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Kind
Length
Subtype
(reserved)
F | m | M | a | A
Encrypted Data ACK (4 or 8 octets, depend on flags)
Encrypted Data Sequence number (4 or 8 octets, depend on flags)
Subflow Sequence Number (4 octets)
Data-Level Length (2 octets)
Checksum (2 octets)

Fig. 9. Secure DSS MPTCP option

Therefore, we propose a simple solution to address the very problem introduced in the MPTCP layer in MPTCP itself — encrypt only the DSN (and DSAck) in MPTCP headers, as shown in Figure 9. As will be detailed later, we can simply combine the keys exchanged on multiple paths (similar to the idea in backup flag attack prevention) to encrypt and decrypt the GSN (symmetric key encryption). The symmetric key can be used in any standard symmetric key encryption algorithms such as AES [26] (e.g., in CTR mode to eliminate the fixed block size [27]). Note that the attacker might have some knowledge about the plaintext GSN which constitutes a "known-plaintext attack" (i.e. knowing a pair of corresponding plaintext and ciphertext). However, modern symmetric encryption algorithms such as AES is resistant to such attacks [26].

However, the challenge here is that, unlike the authentication token that is sent sporadically, GSN is embedded in every data packet. Once a key has been agreed on between the sender and receiver, it is not trivial to update the keys on the fly. For instance, when the very first subflow is established, only the keys exchanged on the first path can be used to encrypt the GSN, which is susceptible to man-in-the-middle attacks on the path (as the keys are exchanged in cleartext and the GSN can be decrypted). Later on, when the second subflow is established, to make encryption more secure we can combine the keys on the second path to form the key material for GSN encryption. This is easy for the second subflow as the key material is readily available. However, it is not the case for first subflow as it needs to switch from unencrypted GSNs to encrypted ones. Therefore, it requires additional signaling mechanism to switch to the newer key material.

Choice of key material change signaling. To address this problem, we can make use of subflow setup or the unused header fields in MPTCP DSS option or `MP_JOIN` TCP option. For example, when a new subflow is setup, the `MP_JOIN` option or first DSS MPTCP option, on the new path, can be used as a signal to start the encryption of the first subflow's GSN without the need of sending any explicit signal. A simple strategy is to start the encryption of GSN on the first subflow upon seeing `MP_JOIN` packet of the second subflow, or upon receiving the first DSS packet on the second subflow. Such an implicit signaling may cause ambiguity for packets that are in-flight together with the `MP_JOIN` packet (unclear what key material to use to decrypt them). To avoid such confusion, we encode an explicit signal in the `MP_JOIN` message of the second subflow, indicating when exactly the encryption should start on the first subflow. This way, as long as the specified DSS is larger than any of the outstanding packets, the receiver knows exactly when to start decrypting the DSS. Note that the signaled DSS needs to be part of the HMAC input during `MP_JOIN` to prevent tampering (as we will discuss later in §V-D). We can add a 4-byte or 8-byte sequence number

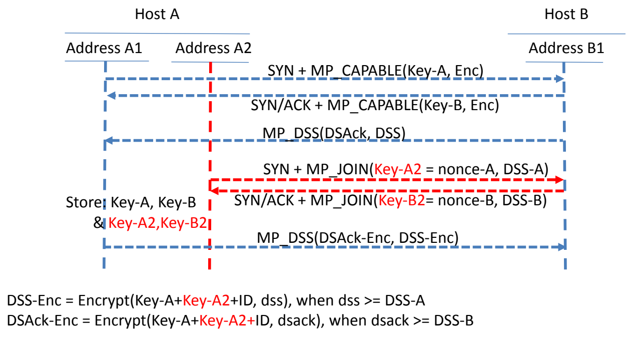

Host A
Host B
Address A1
Address A2
Address B1
SYN + MP_CAPABLE(Key-A, Enc)
SYN/ACK + MP_CAPABLE(Key-B, Enc)
MP_DSS(DSAck, DSS)
SYN + MP_JOIN(Key-A2 = nonce-A, DSS-A)
SYN/ACK + MP_JOIN(Key-B2= nonce-B, DSS-B)
Store Key-A, Key-B & Key-A2,Key-B2
MP_DSS(DSAck-Enc, DSS-Enc)
DSS-Enc = Encrypt(Key-A+Key-A2+ID, dss), when dss >= DSS-A
DSAck-Enc = Encrypt(Key-A+Key-A2+ID, dsack), when dsack >= DSS-B

Fig. 10. Explicit Signaling of GSN Encryption on the First Subflow.

to the `MP_JOIN` packet, and set the packet `Length` field to 16 or 20 respectively. We assume that the signal has to be initiated from the client to avoid race conditions where both sides simultaneously initiate new subflows. When the server acknowledges the message, a reverse signal can be piggybacked in the response `MP_JOIN` to decide when to perform GSN encryption in the reverse direction. We outline this process in Figure 10.

For backward compatibility, we propose using a flag in `MP_CAPABLE` to signal if GSN encryption is supported by both the client and server. If either the client or server does not support encryption, GSN encryption will not be employed.

Choice of secret splitting. As described earlier, we want to combine keys of multiple paths to form the key material for GSN encryption. For example, for an MPTCP connection with 2 subflows, the keys of path$_1$, and path$_2$, can be used for encrypting GSN on path$_1$ and path$_2$ as well. Similarly, for an MPTCP connection with 3 subflows, the keys of path$_1$ and path$_3$ can be used to encrypt GSN on path$_3$. The strategy is to combine the keys exchanged on the primary subflow and the ones changed on the current subflow where GSN encryption takes place. For more security, an alternative strategy could be to combine keys of all active subflows. For example, for an MPTCP connection with 3 subflows, we can use the keys of path$_1$, path$_2$, and path$_3$ for encrypting GSN on path$_1$. However, it may need frequent updates to the keys generated for the paths as more subflows join or leave the MPTCP connection. For example, the key material for the first subflow will be updated twice as subflow 2 and 3 are established; it gets tricky when both new subflows are being established simultaneously because the client and server may not have a consistent view on which of these two subflows gets established first (and may cause a different key material to be formed on the two ends).

Choice of key generation. Based on the above discussion, we propose combining the keys exchanged on only two subflows: (1) the current subflow and (2) the primary subflow. For example, for a connection between hosts A and B, with two subflows 1 and 2, the encryption key of subflow 2 (encryption at A and decryption at B) can be generated as $EncKey_{2_A} = key_A + key_{A2}$ (as shown in Figure 10). The encryption starts as soon as the original value is greater than the signaled DSS value ($DSS - A$ in the example). Similarly, the encryption of the DSAck starts when its value becomes greater than the signaled DSS value in the opposite direction ($DSS - B$ in the example). We omit the details for the reverse direction where the encryption key will be $EncKey_{2_B} = key_B + key_{B2}$. Given that all such numbers are randomly generated, we can simply concatenate them to form the key.

Example scenario. Let us consider a 2-path MPTCP connection setup in Figure 10 to understand how an MPTCP connection progresses when GSN encryption is enabled. All subflows are setup and authenticated as per the original MPTCP connection setup. The required keying material is exchanged in `MP_CAPABLE` (*i.e.* the keys, $Key - A$ and $Key - B$) and `MP_JOIN` (*i.e.* $nonce - A$ and $nonce - B$) as described before. At the end of subflows setup, all the required keying material for the encryption have been exchanged. Moreover, during `MP_CAPABLE` exchange, hosts also negotiate if encryption is supported (Using Enc bit) by both the hosts A and B. Moreover, the `MP_JOIN` option on subflow 2 also carries the signal DSS after which the encryption starts at host A (denoted as $DSS - A$). Similarly, host B informs the host A of the DSS after which the encryption starts at host B (denoted as $DSS - B$). Next, the keying material is combined to form the encryption keys ($DSS - Enc$ and $DSAck - Enc$) for each subflow. When data transmission starts, GSNs are encrypted on path 2 from the beginning because the key materials are ready (details are omitted in the figure). For encrypting GSNs (and GSN-ACKs) on path$_1$, it needs to start encrypting the GSN at a later point. Note that some data packets might be transferred on subflow 1 before subflow 2 setup is completed. GSNs of these few packets will not be encrypted and visible to the attacker on path$_1$, as shown in Figure 10. However, as soon as subflow 2 setup completes, the encryption can start on path 1 as indicated by "$DSS - Enc$" in Figure 10, so long as $DSS > DSS - A$. Similarly, the encryption of $DSACK$ can start as soon as $DSAck > DSS - B$.

D. Security Analysis

We now discuss the potential attacks that can be launched against our security improvements and show we do not introduce any new vulnerabilities. We follow the same threat model as defined in where a MITM attacker is on one or more communication paths and can both passively eavesdrop on and actively tamper with the subflow(s).

First, more specifically, when an attacker is capable of tampering the first subflow, the whole MPTCP connection can be forced to revert back to a regular TCP connection and thus subject to total control of the attacker. Our proposed changes do not change this result. However, if the attacker is capable of tampering subflows other than the first, our changes will prevent the two classes of cross-path attacks that were possible without the changes.

Second, an active attacker may tamper with a subset of the secrets shared (when the corresponding paths are controlled by the attacker), in which case the wrong key material may be used for authentication token and GSN encryption. The HMAC mechanism introduced earlier during subflow setup already mitigates this threat. For instance, if the key on the second subflow is tampered, the attacker also needs to know the key exchanged on the first subflow to be able to compute an updated HMAC (as the key is part of the input). Even if the attacker can indeed successfully cause wrong keys to be used in a particular subflow, the worst case scenario is denial-of-service — the authentication token generated by either end will be considered invalid — the encrypted GSN may be decrypted using the wrong key material. However, we argue that such attacks are no worse than before, as both are possible if an attacker is on-path.

Third, we consider two possible targeted attacks against GSN encryption:

• Attacking the MPTCP option to indicate the support of GSN encryption in `MP_CAPABLE` messages. An attacker has to tamper with the first subflow to be able to strip such option or fake such an option to either disable GSN encryption or cause confusion later on. However, as we discussed, if an attacker is already capable of tampering with the fist subflow, then complete control or denial-of-service can be easily achieved.

• Attacking the signal carried in an `MP_JOIN` message (e.g., second subflow setup) indicating the GSN of the first subflow after which GSN encryption will begin. An attacker capable of tampering with this subflow can strip or modify the sequence number. However, the HMAC of `MP_JOIN` includes the signaled GSN as input so any tampering will be immediately detected, unless a powerful attacker also eavesdrop the first subflow and know the initial keys exchanged there to be able to compute the updated HMAC. Even for a powerful attacker, in the worst case it is again DoS which is no worse than before.

Finally, we want to ensure that *no unnecessary security features are added in reference to TCP*. Since MPTCP and TCP both reside in the same transport layer, whatever security guarantees not offered by TCP do not have to be offered by MPTCP either. Otherwise, it is likely that the proposed features will be redundant with higher layer ones (*e.g.* SSL/TLS). Specifically, TCP does not provide any confidentiality or integrity of any connection data. The proposed changes to MPTCP do not attempt to provide confidentiality or integrity of any subflow either (*e.g.* a MITM attacker can read/modify the payload or any other fields in an MPTCP packet). What we do address are the security flaws that manifest as cross-path vulnerabilities, where a MITM attacker on one path can infer or tamper with the subflow on another path, which effectively makes MPTCP much more susceptible to MITM attacks — if one of the subflows of an MPTCP connection is under a MITM attacker, the whole MPTCP connection may be subject to attacks. We consider this an increased attack surface for MPTCP and therefore makes it less secure than TCP (as TCP traverses only a single path and is "less likely" subject to MITM attacks).

E. Linux Implementation

We implement a simple version of the proposed countermeasures in Linux kernel 4.3 using MPTCP implementation v0.91 [18]. We add a fixed 8-byte badge field in `MP_PRIO` option to prevent backup flag attack. Our current implementation supports both the existing `MP_PRIO` option formats and the proposed badge field. The option field `Length` can be changed to choose among the desired format. For the length control, we add a new parameter `MPTCP_SUB_LEN_PRIO_B`. For our current implementation, we used a static badge value and have not implemented the secure badge generation, as discussed above. Upon receiving `MP_PRIO` packet with length field `MPTCP_SUB_LEN_PRIO_B`, correct ADDR_ID and badge, the receiver sets the corresponding path as backup, else it silently ignores the request. We modify the required logic in `mptcp_parse_options` functions in *mptcp_input.c* We check the badge field and then verify the token in the `mptcp_handle_options` function.

We encrypt data sequence numbers in `MP_DSS` option to prevent throughput inference attack. At the sender side, for an outgoing packet with `MP_DSS` option, we encrypt the DSN using encryption key. For this, we modify the `mptcp_options_write` function in *mptcp_output.c* file and update the `mptcp_write_dss_data_ack` and `mptcp_write_dss_data_seq` functions. At the receiver end, upon receiving packet with `MP_DSS` option, we first decode the DSN numbers before further processing the packets. We modify the required logic in `mptcp_parse_options` functions in *mptcp_input.c*. We do not keep the encrypted DSNs at the endhosts as DSN is encrypted just before sending packets out in the network and is immediately decrypted upon receiving packet at the destination. In our current implementation, we assume static key for encryption (xor of DSN and key) and assume both the sender and receiver have this key. We leave the exact implementation of key generation and encryption as future work.

Evaluation: To evaluate, we repeat attack experiments and observe that proposed countermeasures prevent the connection hijack attacks by making it impossible to infer connection throughput and use backup vulnerability. We omit the results for brevity. We also evaluate the overhead of proposed modifications on the MPTCP connection throughput. First, we observe that adding a token field to the `MP_PRIO` does not effect the throughput of MPTCP connection, as `MP_PRIO` message is sent only once or very few times over the connection. Second, in our current implementation, the per-packet encryption of DSN can incur processing overhead and may decrease the overall connection throughput. To evaluate this, we evaluate MPTCP connection throughput between two Linux machines with 1 Gbps ethernet cards. We observe very minimal MPTCP connection throughput variation. For example, we observe a throughput of 912 Mbps without encryption scheme, and a throughput of 907 Mbps with GSN encryption. This shows that even at high speed packet transmission the computation overhead of encryption is minimal and does not effect connection significantly. Note that, the connection speed in real scenarios (*e.g.* cellular networks) is much less, therefore adding encryption will not hurt the overall connection throughput.

VI. Conclusions

MPTCP has already been widely deployed (including Android and iOS); however, its security is not well understood. In this paper, we describe MPTCP vulnerabilities that can be exploited to divert traffic and hijack MPTCP subflows. We implemented and evaluated our attacks on a testbed using the MPTCP Linux kernel (v0.91) implementation. We also proposed lightweight encryption-based countermeasures based on secret splitting to mitigate these attacks. Our countermeasures do not introduce any new security flaws to MPTCP. We plan to explore opportunities to leverage the secret splitting as a way to bootstrap keys for higher level protocols such as TLS.

Acknowledgment

Research was sponsored by National Science Foundation under grant #1528114. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation or the U.S. Government. We would like to thank shepherd Olivier Bonaventure for his valuable feedback and suggestions to improve this work.

References

[1] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 6824.

[2] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *USENIX NSDI*, 2011.

[3] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How hard can it be? Designing and implementing a deployable Multipath TCP," in *USENIX NSDI*, 2012.

[4] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J.-Y. L. Boudec, "MPTCP is not pareto-optimal: performance issues and a possible solution," in *ACM CoNEXT*, 2012.

[5] Q. Peng, A. Walid, and S. H. Low, "Multipath TCP algorithms: theory and design," in *ACM SIGMETRICS*, 2013, pp. 305–316.

[6] M. Scharf and A. Ford, "Multipath TCP (MPTCP) Application Interface Considerations," RFC 6897.

[7] "iOS: Multipath TCP Support in iOS 7," http://support.apple.com/kb/HT5977, Jan. 2014.

[8] "MPTCP on Android Devices," http://multipath-tcp.org/pmwiki.php/Users/Android, January 2014.

[9] M. Bagnulo, "Threat analysis for TCP extensions for multipath operation with multiple addresses," RFC 6181, IETF, March 2011.

[10] M. Bagnulo, C. Paasch, F. Gont, O. Bonaventure, and C. Raiciu, "Analysis of Residual Threats and Possible Fixes for Multipath TCP (MPTCP)," RFC 7430.

[11] "Linux Blind TCP Spoofing Vulnerability," The U.S. Department of Energy, Computer Incident Advisory Capability, March 1999.

[12] lkm, "Blind TCP/IP hijacking is still alive," 2007.

[13] Z. Qian, Z. M. Mao, and Y. Xie, "Collaborated off-path tcp sequence number inference attack — how to crack sequence number under a second," in *CCS*, 2012.

[14] Z. Qian and Z. M. Mao, "Off-Path TCP Sequence Number Inference Attack - How Firewall Middleboxes Reduce Security," in *SP*, 2012.

[15] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," in *SP*. IEEE, 2010, pp. 191–206.

[16] Y. Sheffer, R. Holz, and P. Saint-Andre, "Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)," RFC 7454, Internet Engineering Task Force (IETF), 2015.

[17] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt, "Drown: Breaking tls using sslv2," in *USENIX Security 16*, 2016.

[18] C. Paasch, S. Barré *et al.*, "Multipath tcp in the linux kernel," *Available from {www. multipath-tcp. org.}*, 2013.

[19] M. Z. Shafiq, F. Le, M. Srivatsa, and A. X. Liu, "Cross-path inference attacks on multipath tcp," in *HotNets-XII*, 2013.

[20] M. Popovici and C. Raiciu, "Exploiting multipath congestion control for fun and profit," in *Hot Topics in Networks*. ACM, 2016, pp. 141–147.

[21] M. Jadin, G. Tihon, O. Pereira, and O. Bonaventure, "Securing multipath tcp: Design & implementation," in *IEEE INFOCOM 2017*, 2017.

[22] C. Pearce and S. Zeadally, "Ancillary Impacts of Multipath TCP on Current and Future Network Security," *Internet Computing*, 2015.

[23] J. Ma, F. Le, A. Russo, and J. Lobo, "Detecting Distributed Signature-based Intrusion: The Case of Multi-Path Routing Attacks," in *IEEE INFOCOM*, 2015.

[24] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 6824bis-09.

[25] S. Frankel and S. Krishnan, "IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap," RFC 6071.

[26] J. Nechvatal, E. Barker, L. Bassham, W. Burr, and M. Dworkin, "Report on the development of advanced encryption standard (aes)," 2000.

[27] H. Lipmaa, P. Rogaway, and D. Wagner, "Comments to nist concerning aes modes of operations: Ctr-mode encryption," NIST, Tech. Rep., 2000.