Securing MPTCP Connections: A Solution for Distributed NIDS Environments

João Pedro Meira
HASLab, INESC TEC
Universidade do Minho
email: joao.p.meira@inesctec.pt

Rui Pedro C. Monteiro
HASLab, INESC TEC
Universidade do Minho
email: rui.p.monteiro@inesctec.pt

João Marco C. Silva
HASLab, INESC TEC
Universidade do Minho
email: joao.marco@inesctec.pt

***Abstract*—With continuous technological advancement, multihomed devices are becoming common. They can connect simultaneously to multiple networks through different interfaces. However, since TCP sessions are bound to one interface per device, it hampers applications from taking advantage of all the available connected networks. This has been solved by MPTCP, introduced as a seamless extension to TCP, allowing more reliable sessions and enhanced throughput. However, MPTCP comes with an inherent risk, as it becomes easier to fragment attacks towards evading NIDS. This paper presents a study of how MPTCP can be used to evade NIDS through simple cross-path attacks. It also introduces tools to facilitate assessing MPTCP-based services in diverse network topologies using an emulation environment. Finally, a new solution is proposed to prevent cross-path attacks through uncoordinated networks. This solution consists of a host-level plugin that allows MPTCP sessions only through trusted networks, even in the presence of a NAT.**

***Index Terms*—MPTCP, NIDS Evasion**

I. Introduction

Although being at the core of network communications for decades, the regular Transport Control Protocol (TCP) [1] cannot take advantage of nowadays commonly multihomed and multiaddressed hosts. While devices supporting multiple interfaces can be simultaneously connected to multiple networks, TCP connections are restricted to a single transport path between peers.

Enabling transport connections to operate across multiple and potentially disjoint paths, Multipath TCP (MPTCP) extends TCP aiming to enhance resource utilisation, connection resilience, and throughput [2]. It operates by demuxing TCP sessions into multiple subflows while providing a standard interface to applications. Each subflow is an independent TCP session established after the three-way handshake from the principal flow. This means that different TCP connections can be used simultaneously and considered part of the same connection through a locally unique identifier (*Connection ID*).

Despite its usefulness, it presents a potential security risk to networks since Network-based Intrusion Detection Systems (NIDS) can lose their effectiveness when MPTCP is used to fragment attacks across different paths. In the worst case, no single point in the network can inspect all the traffic, hampering NIDS from identifying ongoing attacks. An example of such a scenario consists of hosts connected to multiple networks with uncoordinated domain administration, such as a mobile device simultaneously connected to a private network and a 4G/5G provider.

Considering the increasing number of operating systems supporting the protocol extension and services taking advantage of its features [3], it is of utmost importance to conceive and develop suitable protection mechanisms. Within this context, the contributions of this work are threefold:

i) a tool for easy configuration of simulated MPTCP enabled topologies. This not only allows for assessing security aspects of MPTCP but also for deploying and testing generic applications running over the protocol;
ii) a detailed real-world demonstration of how MPTCP can be exploited to conduct cross-path data fragmentation attacks undetected by NIDS;
iii) a solution to the demonstrated issue that consists of a plugin to Intel's *mptcpd* daemon, which is the current primary way to manage MPTCP connections in the user space of Linux-based operating systems.

This paper is organised as follows: Section II characterises the security-related issues of using NIDS in MPTCP-enabled environments while presenting the most effective solution addressing different scenarios; Section III presents a tool developed to easily study MPTCP-enabled environments. Based on this, a test scenario is deployed to demonstrate that solutions up to this point cannot avoid NIDS evasion when devices are connected through uncoordinated networks; Section IV introduces a host-based solution designed to prevent MPTCP connections from being established through untrusted networks; Finally, Section V brings the final remarks of using the proposed solution.

II. Related work

Security implications of MPTCP have been a concern since its first specification [4]. Some of them are related to vulnerabilities in protocol implementations [5], [6] and have received attention within the current version [2]. The primary source of concerns, though, arises from the fact that no longer a connection can be interpreted as a single data stream, which breaks many of the TCP assumptions with implications to NIDS operation and consequently to network security [7]. This characteristic leads to two new scenarios for cross-path data fragmentation attacks.

The first scenario was described by Foster [8] in which an attacker takes advantage of NIDS unable to identify multiple subflows as a unique MPTCP connection. As illustrated in Figure 1, this type of attack can be performed even against hosts with a single interface since it supports MPTCP. Here, the attacker forces data fragmentation of attack payloads across different subflows, aiming to evade detection at the network level. Data will be multiplexed at the host's transport layer and delivered undetected to the application. Foster also proposed a couple of solutions to this issue [8]. The first is a proxy that reassembles MPTCP subflows back to TCP streams before analysis. The second one extends a NIDS to understand MPTCP connections, reassembling subflows by itself.

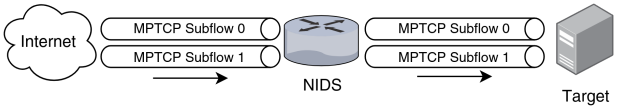

Fig. 1: Single NIDS

The other scenario in which data fragmentation represents security issues relates to distributed NIDS environments. As presented in Figure 2, data from the same MPTCP connection can arrive through different and independent networks, and even if both networks are monitored by a NIDS, none of them has a complete picture of traffic content. Within this context, the problem solution is more complex and depends on the administrative control of each network and NIDS. When all detection systems (e.g., NIDS 1 and NIDS 2 in Figure 2) are controlled by the same entity, it requires coordination mechanisms. Barksdale [9] addressed this scenario by proposing a cooperative approach in which distributed NIDSs redirect MPTCP traffic to a previously selected primary NIDS. Then, the second solution in [8] can be applied to reassemble the traffic.

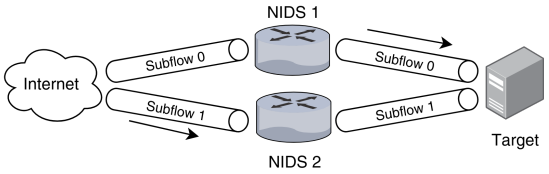

Fig. 2: Distributed NIDS

Although coordinated NIDS are generally effective, typically, multihomed devices are connected through networks operated by different providers, hampering coordination strategies. Therefore, this is a scenario where a generic strategy for protecting networks from cross-path data fragmentation attacks in MPTCP is an open issue addressed in this work.

III. Cross-path attacks to MPTCP

This section presents two of the contributions in this work. Firstly, it describes a tool developed to simplify the deployment of experimental MPTCP-enabled environments based on the network emulator *CORE* (see Section III-A). Then, a test scenario is deployed to demonstrate how MPTCP can be used to evade NIDS in distributed environments (see Section III-B). The results show that although its increasing support, MPTCP still can be used for low complex and successful attacks (see Section III-C).

A. Experimental tool

Even with MPTCP enabled in the Linux kernel by default, it does not work *out of the box*, requiring multiple manual configurations for each interface in a multihomed system. Typically, these configurations represent a lengthy process that scales with the number of network interfaces.

To handle these tasks in an experimental environment, this work introduces a tool to rapidly configure simulated MPTCP-enabled topologies. This consists of a Python-written *parser* for *CORE* emulator responsible for taking a *TOML* input configuration file and automatically deploying a topology with multihomed devices configured to use MPTCP. An important aspect of this approach is that an equal session is always created for the same configuration file, allowing for consistency between tests. This tool and its documentation are publicly available. [1]

B. Test environment and methodology

Based on the simulation platform described in Section III-A, a test topology is deployed to analyse NIDS's effectiveness against cross-path attacks in distributed and non-coordinated environments. The analysis considers two of the most used open-source NIDS, namely, *Snort3*[2] and *Suricata*[3]. The tested topology corresponds to the second scenario in Section II and consists of NIDS independently analysing partial traffic traversing different networks (see Figure 2).

Both *Snort3* and *Suricata* work by inspecting data stream content against predefined rules to identify possible threats. These rules specify patterns that trigger a correspondent alert when detected within the analysed flow. In the MPTCP scenario considered for this work, an adversary can fragment an attack payload through multiple subflows to evade NIDS's anomaly identification processes.

Two baseline scenarios are also deployed to avoid false-positives when analysing NIDS effectiveness. The first one (*Base 1*) aims at evaluating whether they are capable of detecting attacks when data is fragmented in single flow TCP sessions (i.e., without MPTCP). The objective is to identify NIDS's ability to reassemble fragmented data before applying their detection processes. The second scenario (*Base 2*) consists of evaluating whether the new option added to the TCP header (i.e., value 30) can affect NIDS effectiveness, as it can change the signature used to detect traffic anomalies. The baseline scenarios are analysed using just one path between the attacker and the target.

[1] https://github.com/MPTCP-Lab/mptcp_test_bed
[2] https://www.snort.org/
[3] https://suricata.io/

The test methodology consists of arbitrarily fragmenting HTTP-based attack payloads injected in the simulated environment across MPTCP subflows in both links of Figure 2. To do so, a script provokes link failures forcing part of the traffic to be transmitted through the second link. This approach does not require crafting IP packets and can be easily replicated in real scenarios with similar results. The effectiveness of both NIDS is evaluated by comparing the number and priority (Pt.) of the alerts triggered in baseline and test scenarios. Alert's priority indicates the severity of the anomaly detected. They vary between 1 and 3 in *Suricata* and *Snort3*, where lower priority values indicate more severe events.

A realistic test case was deployed to evaluate the impact of these attacks. It uses the database of the vulnerability scanner *Nikto2*[4] to inject anomalous traffic within the tested scenarios along with the fragmentation strategy previously described. This database consists of thousands of HTTP requests crafted to look for possible vulnerabilities in web servers.

C. Test results

Comparing the results from different scenarios shows that triggered alerts' number and priority differ significantly. For the highest alert severity (i.e., priority 1), both *Suricata* and *Snort3* (see Figure 3) identify a much lower number of possible attacks in the distributed test scenario. In the case of *Snort3*, it only managed to detect 2% of the threats when compared with the *Baseline 1*. Similar results are observed for priority 2 alerts.

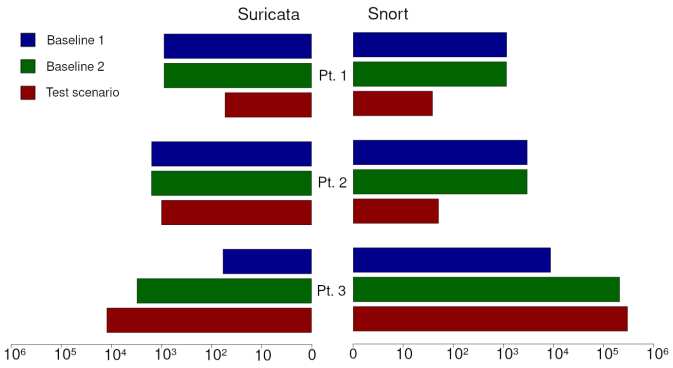

Fig. 3: Number of alerts

Even though alert priority can give a good overview of the severity of detected threats, it is also interesting to analyse the triggered rules and how they vary in the different scenarios. As exemplified in Table I, for *"Cross-Site Scripting Attempts"*, both NIDS managed to detect around 300 occurrences when inspecting traffic in the baseline scenarios. However, when fragmenting the traffic across multiple MPTCP subflows, these occurrences dropped to just 2, representing around 1% of the initial amount.

Contrary to priority 1 and 2, the number of alerts with priority 3 rises substantially, leading to an overall increase in the total number of alerts reported by both NIDS in the distributed scenario. However, these alerts are mostly false positives. What happens is that both NIDS fail to detect the real threats and end up triggering other alerts that are not accurate and typically have a lower priority.

TABLE I: Example of alerts per scenario

Alert description and Priority (Pt.)	Scenario		
	Base 1	Base 2	Test
Suricata			
Possible cross-site scripting attempt [Pt.1]	299	299	2
GLP WEB_SERVER /etc/passwd [Pt.2]	268	268	48
Can't match response to request [Pt.3]	0	2950	5748
Snort3			
Likely cross-site scripting attempt [Pt.1]	284	284	2
SERVER-WEBAPP /etc/passwd [Pt.2]	267	267	0
Response before client request [Pt.3]	2	2	6912

Although the results presented for the test scenario are the combination of both NIDS running on different networks, most false-positive alerts occur in the NIDS connected to the main subflow network. Therefore, for a real-life scenario, depending on how the attack is conducted, one of the networks will probably not have access to any hint of a possible threat. This aspect ratifies the relevance of providing network administrators with mechanisms to manage MPTCP connections within their administrative domains.

The entirety of the results is also publicly available[5], and the tests conducted can easily be reproduced using the tool described in Section III-A.

IV. Proposed solution

Although some efforts have tried to secure MPTCP-enabled environments, Section III demonstrates the protocol still can be easily used to evade NIDS by fragmenting payload attacks through different subflows. This is particularly critical in scenarios where subflows are established in networks with no coordinated control. Up to this point, the proposed strategies relied on processing traffic at each network entry point. In this case, none of the NIDS can inspect the entire flow looking for possible threats. Neither the detection system in one network has information about the remaining networks to which a device is connected.

Instead of solving the problem at the network level, this work introduces a security-focused plugin designed to address the issue at the host level. This solution extends the *mptcpd* with a mechanism that only allows MPTCP sessions through trusted networks, which can avoid partial traffic arriving through uncontrolled networks. The proposed solution works even in the presence of Network Address Translation (NAT) and is publicly available[6] to be used either in the simulated environment (see Section III) or in production systems.

As of version 0.8 of *mptpcd* (the current version at the time of writing this paper), developing a security extension that preserves the modularity and extendability purposes of

[4]https://cirt.net/Nikto2

[5]https://github.com/MPTCP-Lab/mptcp-nids-evasion-tests

[6]https://github.com/MPTCP-Lab/net_check_plugin

using plugins is a challenging task since, *mptcpd* does not provide the necessary mechanisms to such logic. Thus, some modifications to *mptcpd* were required.[7]

A. Plugin

Securing MPTCP connections at the host level consists of controlling which networks a device can use for multipath sessions. To do so, the proposed plugin[6] uses either an allowlist or blocklist approach, indicating the trusted and untrusted networks, respectively. For devices using private IPv4 addresses, the plugin can use the Session Traversal Utilities for NAT (STUN) [10] to identify the public address of its NAT server [11][12] and, consequently, the network to which it is connected before establishing an MPTCP session.

Each time the plugin is notified about a new IP address for one of the device's interfaces, it checks whether the underlying network is on the allowed or blocked list. If the new address belongs to an untrusted network, the plugin halts the event propagation and generates a new flow to remove the previous interface's addresses (simulating a process typically triggered by the kernel). Then, it uses NetFilter[8] to set firewall rules blocking MPTCP communications through that interface.

On the other hand, if the new address belongs to a trusted network, the proposed plugin allows the event propagation to lower priority plugins. It also removes any firewall rule previously set to that interface, allowing MPTCP communications. Figure 4 details the plugin operation when a new address event is received.

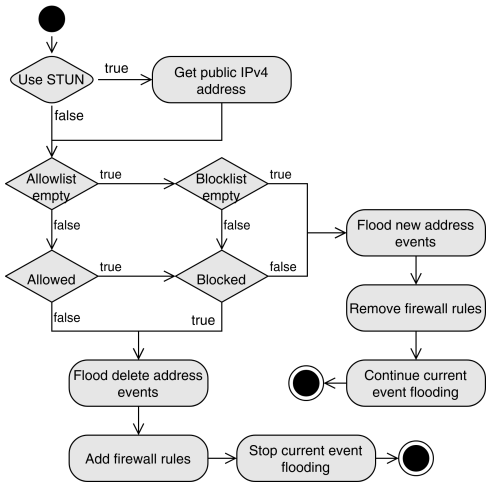

Fig. 4: New address event handling

Although the plugin prevents MPTCP communication through the interfaces connected to untrusted networks, it actually does not block the connection establishment itself. Since MPTCP falls back to ordinary TCP when one of the communication sides does not support it [2], the plugin simply takes advantage of it by removing the MPTCP option from *SYN* packets. In this way, ordinary TCP sessions still can be established through untrusted networks.

[7]https://github.com/MPTCP-Lab/mptcpd/tree/patched_version

[8]https://www.netfilter.org/

V. Final remarks and future work

Since the proposed plugin strips down the MPTCP options, in a new experimental analysis based on *Test scenario* in which the target uses the proposed solution, an attempt by an attacker to initiate an MPTCP communication through an untrusted network will fall back to plain TCP, resulting in an alerts distribution similar to the *Baseline 1*. If both networks are trustworthy, MPTCP communication will be established as expected, allowing to take advantage of its benefits. Moreover, as the plugin does not prevent communication through untrusted networks, it does not affect applications' functionalities or QoS.

Notice the proposed solution does not entirely replace those presented in Section II. Instead, it covers an open issue while enhancing how users protect and manage MPTCP connections, allowing for total control of when and where the protocol can be used. Thus, in future work, integrating this solution with a coordinated NIDS approach can provide a unique mechanism towards securing MPTCP-base communications.

Acknowledgements

This work is financed by National Funds through the FCT - Fundação para a Ciência e a Tecnologia, I.P. (Portuguese Foundation for Science and Technology) within the project FLEXCOMM, with reference EXPL/CCI-INF/1543/2021.

References

[1] "Transmission Control Protocol." RFC 793, Sept. 1981.

[2] A. Ford, C. Raiciu, M. J. Handley, O. Bonaventure, and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses." RFC 8684, Mar. 2020.

[3] F. Aschenbrenner, T. Shreedhar, O. Gasser, N. Mohan, and J. Ott, "From single lane to highways: Analyzing the adoption of multipath tcp in the internet," in *2021 IFIP Networking Conference (IFIP Networking)*, pp. 1–9, IEEE, 2021.

[4] M. Bagnulo, "Threat Analysis for TCP Extensions for Multipath Operation with Multiple Addresses." RFC 6181, Mar. 2011.

[5] F. Demaria, "Security evaluation of multipath tcp : Analyzing and fixing multipath tcp vulnerabilities, contributing to the linus kernel implementation of the new version of the protocol," 2016.

[6] V. A. Kumar and D. Das, "Data sequence signal manipulation in multipath tcp (mptcp): The vulnerability, attack and its detection," *Computers & Security*, vol. 103, p. 102180, 2021.

[7] C. Pearce, "Multipath tcp, pwning today's networks with tomorrow's protocols," tech. rep., Blackhat, 2014.

[8] H. A. Foster, "Why does mptcp have to make things so complicated?: cross-path nids evasion and countermeasures," 2016-09.

[9] I. Barksdale, Warren L., "A cooperative ids approach against mptcp attacks," 2017-06.

[10] M. Petit-Huguenin, G. Salgueiro, J. Rosenberg, D. Wing, R. Mahy, and P. Matthews, "Session Traversal Utilities for NAT (STUN)." RFC 8489, Feb. 2020.

[11] M. Holdrege and P. Srisuresh, "IP Network Address Translator (NAT) Terminology and Considerations." RFC 2663, Aug. 1999.

[12] K. B. Egevang and P. Srisuresh, "Traditional IP Network Address Translator (Traditional NAT)." RFC 3022, Jan. 2001.