Multipath TCP IDS Evasion and Mitigation

Zeeshan Afzal(✉) and Stefan Lindskog

Karlstad University, Karlstad, Sweden
{zeeshan.afzal,stefan.lindskog}@kau.se

Abstract. The existing network security infrastructure is not ready for future protocols such as Multipath TCP (MPTCP). The outcome is that middleboxes are configured to block such protocols. This paper studies the security risk that arises if future protocols are used over unaware infrastructures. In particular, the practicality and severity of cross-path fragmentation attacks utilizing MPTCP against the signature-matching capability of the Snort intrusion detection system (IDS) is investigated. Results reveal that the attack is realistic and opens the possibility to evade any signature-based IDS. To mitigate the attack, a solution is also proposed in the form of the *MPTCP Linker* tool. The work outlines the importance of MPTCP support in future network security middleboxes.

Keywords: IDS evasion · Multipath transfers · TCP · Snort · Middleboxes

1 Introduction

The single path nature of traditional TCP[1] is arguably its greatest weakness. Traditional TCP implements connections between two sockets (pairs of host IP addresses and port numbers) and this 4-tuple needs to remain constant during the lifetime of the connection. Today, end-hosts are equipped with multiple network interfaces, all of which can have a unique IP address. A traditional TCP connection from/to such a host will be limited to using only one path (defined by the two sockets) at a time. Thus, there is a potential and a need to implement TCP connections between end-hosts that can utilize all possible paths that the hosts provide. Such connections will provide higher *availability* and higher *throughput*, among many other advantages. They can also solve a number of existing problems in the Internet of today [11,29].

Multipath TCP (MPTCP[2]) is an extension to traditional TCP that adds the missing ability and enables the use of multiple paths between hosts. It is carefully designed to work on the Internet of today. It also has a fallback mechanism that allows it to switch to traditional TCP when MPTCP is not feasible. The developers of MPTCP have until now specially focused on the feasibility aspects of the protocol and also ensure that no residual security vulnerabilities

[1] Traditional TCP is the same TCP we know and use today.
[2] Multipath TCP is also referred to as MPTCP.

J. Lopez and C.J. Mitchell (Eds.): ISC 2015, LNCS 9290, pp. 265–282, 2015.
DOI: 10.1007/978-3-319-23318-5_15

exist in the protocol itself. However, the network security impacts of using the new MPTCP protocol on the existing infrastructure are yet to be thoroughly investigated.

1.1 Motivation and Research Questions

Internet hosts a number of middleboxes. These middleboxes are deployed either in the form of dedicated hardware or software-based solutions. Nevertheless, most of them are transparent (implicit) with end-hosts unaware of their existence. These middleboxes are deployed to do more processing than simply forwarding the packets that pass through. More and more enterprises are implementing and deploying middleboxes in the form of load balancers, Network Address Translators (NAT), firewalls and Intrusion Detection and Prevention Systems (IDPS) to optimize performance and enhance network security. The middleboxes that are used to improve security perform intrusion detection and prevention. One widely used technique in such systems is based on pre-defined signatures. The signatures used for detection have been developed over the years by making a number of assumptions about the behavior and pattern of the traffic. As revealed by [23], many of those assumptions may no longer be valid with the advent of new protocols like MPTCP. In fact, MPTCP opens the possibility of intrusion detection system (IDS) evasion, where a sender can fragment the data stream and send the fragments across multiple paths in a way such as to bypass the IDS.

This has left the enterprises with a headache as the middlebox infrastructure is unaware of MPTCP and allowing such traffic to go through might come at the cost of degraded security. An increasing number of enterprises have decided not to take that risk and instead configure their middleboxes to remove the MPTCP option and force the protocol to fall back to traditional TCP. A few years ago, Honda et al. [12] found that 14 percent of the tested paths use middleboxes to eliminate the MPTCP options. This behavior of middleboxes is a stumbling block in the universal deployment of MPTCP. There is therefore a need to investigate whether the concerns related to the use of MPTCP are actually true and how dangerous the potential attacks could be.

IDS evasion using Multipath TCP is possible, as shown in [23], but how do the current IDS solutions react under such an intrusion attempt? What is the severity or seriousness of the situation? Is there a solution to the problem? In this paper, we try to answer these questions. We use *Snort* [24] as an example of a popular software-based middlebox and investigate how cross-path data fragmentation using MPTCP affects its detection capability. We do so by generating attack traffic corresponding to the latest Snort rule set using the tools we have developed. The traffic is fragmented and sent across a varying number of paths using MPTCP to a server where Snort is running (loaded with the same rule set) as a middlebox. The number of intrusions detected by Snort is counted and compared to the benchmark results collected using the same set-up but with traditional TCP (or a single subflow). The goal is to establish the extent to which the detection suffers and security degrades as a consequence of cross-path

fragmentation. Finally, to mitigate the degraded detection, a novel solution is proposed in the form of the *MPTCP Linker* tool.

1.2 Contribution

The key contributions of this paper are as follows: (1) A statistical analysis of the latest Snort rule set (snapshot 2970); (2) Development of novel tools to parse Snort rules, generate relevant payloads, fragment equally across available paths and send them to a server using MPTCP; (3) Development and implementation of a research methodology to test the effects of cross-path data fragmentation using MPTCP on Snort's detection engine; and (4) Introduction of a solution in the form of the MPTCP Linker tool that can be used as an extension to Snort to mitigate the cross-path fragmentation attack.

1.3 Paper Structure

The rest of the paper is structured as follows. Section 2 summarizes related work. Section 3 provides some background on MPTCP and Snort, which is relevant for future sections. Section 4 provides a description of the research methodology developed to carry out the work and the working of different tools. Section 5 presents a statistical analysis on the Snort rule set. Section 6 shows the results of the testing and evaluation of Snort. Section 7 describes a solution to mitigate the attack. Section 8 provides an outlook and, finally, Sect. 9 provides concluding remarks.

2 Related Work

Much effort has been put into the feasibility and functionality side of MPTCP. Honda et al. [12] studied the real world feasibility of extending TCP in their work. They used volunteers across 24 countries to test the traversal of unknown TCP options through the middleboxes deployed in different access networks. In total, 142 paths were tested between September 2010 and April 2011. The results showed that 20 of the 142 paths removed or stripped the unknown MPTCP options, while the remaining 122 paths passed the options intact. Since most paths allowed the unknown option, the authors concluded that extending TCP using new options is feasible as long as the new extension has a fallback mechanism. They also outlined that the paths with middleboxes that strip the options can easily be configured to allow the unknown options to pass through, provided that the new options do not introduce a security risk. Lanley [14] also studied the viability of TCP extensions. The work of both Honda et al. and Lanley has influenced the design of MPTCP in its current form.

Some focus has also been placed on the inherent threats in the MPTCP protocol extension. A draft [6] has been proposed to identify potential vulnerabilities in the MPTCP design. The overall goal is to ensure that MPTCP is no worse than traditional TCP in terms of security. A number of potential attacks have

been identified and their possible solutions have been proposed. Such solutions, if implemented, can help push MPTCP to become a standard.

However, the network security implications of MPTCP have not been studied very much. The most significant work in this regard was that of Pearce and Thomas [23]. In their work, they investigated the effects of MPTCP on current network security and indicated that the existing security infrastructure is not MPTCP aware. To demonstrate the risk, a tool [28] was developed to show preliminary IDS evasion. Our work will further contribute to this area and highlight the network security implications of MPTCP using novel tools and will propose novel solutions.

3 Background

In this section, we discuss some key concepts and information that will be beneficial to comprehend the future sections. First, multipath transfers are discussed with a focus on MPTCP. Network security implications are then emphasized. Finally, the Snort IDS is presented.

3.1 Multipath Networking

TCP has enjoyed success for decades and will continue to do so. However, there have been an ever increasing number of situations where it falls short. The dependency of a TCP connection on the same pair of IP addresses and port numbers throughout the life of the connection is becoming an issue for a number of use cases and applications. Therefore, researchers attempted to address the issue as early as 1995 in the form of a draft [13]. The draft identified different cases where the above mentioned dependency is harmful and proposed modifying TCP and adding a new Protocol Control Block (PCB) parameter. This parameter will allow the IP addresses to change during the course of a connection. Recent efforts have seen the development of the SCTP protocol [26], which has a great deal of potential but has so far failed to achieve wide-scale deployment. There are two main reasons behind the failure of such efforts. First is their revolutionary nature, which requires changes in the software and sometimes even hardware. Second is the feasibility aspect. Any attempt to introduce an almost completely new protocol on the Internet will most likely fail. Such a protocol will not be able to traverse far across the Internet because most of the networking infrastructure on the Internet assumes that TCP and UDP are the only two transport layer protocols that exist. In addition, the proposed solutions had no fallback mechanism, which made them a failure.

In this regard, the latest efforts in multipath networking have more of an evolutionary nature. The feasibility of the solution has been a paramount consideration because, no matter how good a solution is, it is only going to succeed if it will be feasible to use it on a wide scale. The IETF established a working group called Multipath TCP in 2010. The group was tasked to develop mechanisms that can add the capability of multiple paths to the traditional TCP without

requiring any significant modifications to the existing Internet infrastructure. The deployability and usability of the solution were also two key goals. The first draft was put forward by the working group in 2011 in the form of RFC 6824 [9].

Overview of MPTCP. MPTCP is an extension to traditional TCP that enables a TCP connection to operate across multiple paths simultaneously [9]. This brings the support to a number of use cases, which was not possible before. It is designed to run on top of today's Internet infrastructure and has a fallback mechanism that allows it to be backward compatible. Crudely, an MPTCP connection consists of one or more subflows. Each of these subflows is a proper TCP connection but with additional MPTCP options that allow every subflow to be linked to an MPTCP connection. A detailed technical discussion of the protocol is beyond the scope of this work. Hence, we discuss only some key concepts and the operation of MPTCP in the subsequent text.

Implementation. MPTCP is realized using the options field available in the TCP header. IANA has assigned a special TCP option (value 30) to MPTCP. Individual messages use MPTCP option subtypes. MPTCP implementations are already available on a number of operating systems. It is available for Linux [20], BSD [4] and Android [7]. Commercially, Apple has implemented it in iOS7 [5] and OS X Yosemite [22]. In our work, the Linux kernel implementation [20] of MPTCP is used.

Initiating an MPTCP Connection. An MPTCP connection uses the same three-way connection establishment handshake as the traditional TCP but with an *MP_CAPABLE* option attached to all the exchanged messages. This option serves two purposes. First, it announces to the remote host that the sender supports MPTCP. Second, it carries additional information, e.g. random keys, which can be used in forthcoming exchanges. Figure 1 shows the required interaction between a multipath capable client and server to successfully complete the MPTCP handshake. This initial handshake is also called the *MP_CAPABLE* handshake.

Addition of a New Subflow. Additional subflows can be added to an established MPTCP connection. This is achieved in the same way as initiating a new

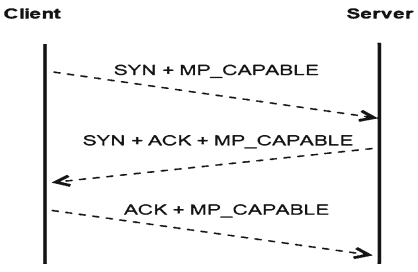

Fig. 1. MP_CAPABLE handshake.

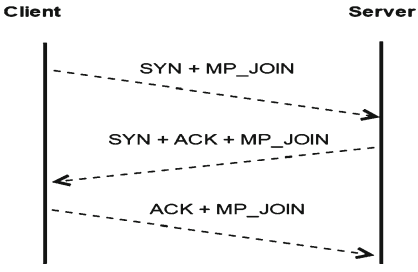

Fig. 2. MP_JOIN handshake.

MPTCP connection but instead making use of the *MP_JOIN* option. This option uses the keys exchanged in the *MP_CAPABLE* handshake to tell the remote end that the connection request is not for a new connection but relates to an existing one. Figure 2 shows the handshake involved. This handshake is also known as *MP_JOIN* handshake. New subflows can be added or removed at any time during the lifetime of a connection. For further details, see [9].

Data Transfer using MPTCP. MPTCP ensures reliable and in-order delivery of the data across all subflows of an MPTCP connection using a Data Sequence Number (DSN). Every subflow has its own transmission window (sequence number space), and the DSS option of MPTCP is used to map the subflow sequence space to the overall MPTCP connection space. This enables data to be retransmitted on different subflows in the event of failure. On the receiver side, MPTCP uses a single receive window across all subflows.

The important thing to note is that the proposed MPTCP standard leaves the exact routing or scheduling of traffic among the subflows up to the implementation. In a common use case where a higher throughput is desired, all available paths (subflows) can be used simultaneously [9]. A sender of the data can tell the receiver how the data are routed among the subflows using the DSS option. The receiver uses this information to re-order the data received over different subflows before passing them on to the application layer in the correct order. Thus, MPTCP enables the sender to choose how to split the input data among the available subflows.

3.2 Network Security Reflections

MPTCP has a number of network security implications. It affects the expectations of other entities in the environment where the protocol extension is used. Network infrastructures can not expect MPTCP to behave in ways similar to those of the traditional TCP.

In this regard, one such observation that is most relevant for this work is *cross-path data fragmentation*. As discussed earlier, MPTCP allows the use of multiple paths simultaneously. A sender can also distribute the data stream among the subflows as it wishes. This opens the possibility to perform *cross-path fragmentation attacks*. A sender can send a known malicious payload by fragmenting it across the subflows in a smart way. The activity will not be detectable by any existing (or non-MPTCP aware) network security middleboxes because, for all they know, every subflow (path) is an independent TCP connection with an unknown fragmented payload.

To further exacerbate the situation, this is just a single problem. The fact that the network paths that are part of the MPTCP connection could be controlled by different Internet Service Providers (ISPs) implies that there may not be any single point in the network that can be used to observe the traffic from all paths. This in turn implies that, even if a middlebox is intelligent enough to know that different subflows make up one MPTCP connection, it may not be able to properly aggregate the traffic and inspect, simply because some subflows may not be visible to it.

3.3 Snort

In the open source world, Snort [24] is the de facto standard IDPS. Snort is effective and is available under the GNU General Public License [25], making it the most widely deployed solution in the world [27].

Snort Operation. Snort provides protection in two ways. It can provide detailed statistics on traffic that can be used for detection of anomalies. It can also provide pattern-matching, which can be used for signature detection. Snort utilizes a rule-based detection approach to perform signature matching on the contents of traffic and detect a variety of attacks [24]. It can currently analyze packets belonging to the four protocols TCP, UDP, ICMP and IP. The detection engine of Snort is configured based on rules. Rules are used to define per packet tests and actions. Once Snort is running with a set of rules, it analyzes every packet that passes through and checks whether the specification given in any of the rules exactly matches the packet. If it detects a match, then it has the possibility to generate and send real time alerts to the *syslog* facility, a UNIX socket or a CSV formatted alert file. The paper by Roesch [24], who is the founder of Snort, provides more detailed information on it.

Rules. Snort rules are written in a simple and flexible, yet powerful, language. Basically, every Snort rule can be divided into two logical parts, a rule *header* and the rule *options*. The rule *header* defines the action of the rule, the protocol it detects, the pair of source IP address/port number, and the pair of destination IP address/port number. The rule *options* consist of the alert message to display and further information about the processing to perform on the packet as well as which parts of the packet should be inspected. The following is an example of a Snort rule.

```
alert tcp any any -> any any (msg:‘‘Sample alert’’;content:‘‘Hello’’;)
```

The above rule should trigger on an incoming TCP packet from any source IP address and port going to any destination IP address and port as long as it contains the text “Hello” in its payload.

4 Experimental Methodology

This section will provide information about the experimental methodology that was developed and was used to carry out the testing. Figure 3 shows the overall set-up. The client and server sides, both, have component modules that work together to achieve the task. The client and server sides are described in the forthcoming text.

4.1 Client Side

The client side of the methodology is made up of the latest Snort rule set and three tools, namely the *Rule Analyzer*, the *Rule Parser* and the *MPTCP tool*. We will describe all of these components in the forthcoming text.

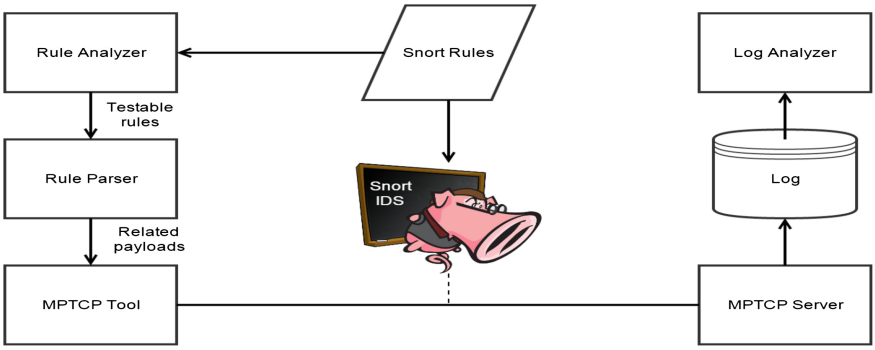

Fig. 3. Experimental setup.

Snort Rules. The latest Snort registered user rule snapshot (2970), available at [27], is used in this work. All the rule files inside the folder (except three[3]) are passed on to the Rule Analyzer.

Rule Analyzer. *Rule Analyzer* performs the following tasks. It reads all input files one by one, analyzes every single rule and then outputs statistics. The statistics provide a distribution of rules by protocol, keyword and other parameters as requested by the user. The Rule Analyzer is used to perform the statistical analysis on the rules.

Rule Parser. A mechanism to parse every Snort rule and craft a consistent payload according to the details in the rule was required. A literature review uncovered tools such as Stick [10], PCP [21] and Mucus [18]. Further investigation found two main drawbacks to these related approaches. First, they are stateless, which means that they can only handle rules relating to stateless protocols such as ICMP and UDP. TCP, on the other hand, maintains states, and any tool not supporting it will be limited in its coverage. Secondly, they are at least ten years old and are thus not compatible with the rule syntax of modern IDSs. A more recent attempt has been the rule2alert [16] tool which, although it is a great improvement over other approaches, still lacks some required functionality. For this work, a further enhancement of [16] was conducted. The tool developed has been presented at [3]. It is able to translate a large number of Snort rules into corresponding payloads. Such payloads can later be encapsulated in packets and used to test the detection accuracy of Snort. This is due to the facts that the payloads are generated directly from the details in the rule and that Snort uses the same rules for detection.

The tool developed searches for the *content* keyword in the *option* field of every rule. The keyword *content* is used to define the signature that Snort should search for in a packet payload. The signature can be a text sequence, binary

[3] Among these three files, one contains old and deleted rules, one is for local rules and the third is for obsolete X windows rules. All of these were deemed irrelevant.

data or even a combination of text and binary data. If Snort detects data that exactly match that given using the *content* keyword in a packet payload, the test is successful and the remainder of rule *options* are checked. The tool uses regular expressions to find the signatures mentioned in every rule (signatures follow the *content* keyword, e.g. *content:"Hello"*) and extracts them. A payload is crafted for each rule and sent to the MPTCP tool, which will be described later.

Snort also supports a number of additional modifier keywords, e.g. *dsize*, *offset*, *distance* and *within* in the rule *options* field. These keywords modify the semantics of the *content* keyword. As an example, the *offset* keyword tells Snort that, instead of starting pattern matching for a signature at the beginning of the payload in a packet, Snort should actually start pattern matching from the given offset value. Hence, if the modifier *keywords* are ignored when crafting the payloads, then it is implausible to expect Snort to generate alerts. The *Rule Parser* tool developed is smart enough to craft the payload by taking modifier *keywords* (if any) into account.

Limitation. It is not currently possible to test all Snort rules. There are a number of keywords that are more complex and require extra effort to build the corresponding payload for. That is why Snort rules are classified into *testable* and *untestable* categories when the statistical analysis of the rules is conducted in Sect. 5.

MPTCP Tool. This is the core tool responsible for generating MPTCP packets with the given payloads and sending them to the destination server. It runs in a virtual machine with Ubuntu as the operating system. The Linux kernel implementation [20] of MPTCP is used, which adds the MPTCP support to the kernel's networking subtree. The tool is inspired by the software [28] released by Pearce et al. [23] with a few improvements.

The tool implements an ad hoc MPTCP scheduler as in [28]. The main criterion used by the tool to fragment the data stream is that the destination should be able to correctly reassemble data in the right order. In that sense, the minimum size of a fragment has to be one byte or two hex digits. The tool attempts to fragment data equally across all available subflows as much as possible. This means that, if a way to split the data equally among all available subflows exists, the tool will do that. However if, for a given set of data, it is not possible to split it equally using all available subflows, the tool will use a subset of available subflows. The tool uses the following formula to calculate packet size for each subflow:

$$pkt\_size = ceiling\left(\frac{length\_of\_data}{available\_subflows}\right)$$

As an example, let us consider a data stream, "netmap" or 0x6e65746d6170 in hex, and two available subflows. The data stream will be fragmented in the following manner. Subflow 1 will be used to send 0x6e6574 and subflow 2 will be used to send 0x6d6170. If the available subflows are three, then subflow 1 will send 0x6e65, subflow 2 send 0x746d and finally subflow 3 send 0x6170. However,

if the available subflows are four, there is no way to divide the data stream equally using all subflows. Thus, one subflow (subflow 4) will not be used at all. Regardless of how data are fragmented, it should be noted that the destination always receives data in a way such that they can be reassembled back to create the original data stream of 0x6e65746d6170 or "netmap".

The ad hoc scheduler used might not be representative of the actual intention of MPTCP developers, but there is nothing to stop an attacker from exploiting things in a way in which they were not intended to be used. In fact, splitting data equally among available paths (subflows) might also not be the most effective way from an attacker's point of view. There could be smarter ways of fragmenting data across subflows while still ensuring that the receiver gets the original data stream. We believe that the scheduler implemented is good enough to show the scope of the problem. The improved version of our tool can perform the following tasks:

1. Test the server for MPTCP support.
2. Perform a three-way MPTCP handshake with the MPTCP server.
3. Add the user defined number of subflows to the connection.
4. Send the given payload (can be text, binary or both) to the server using any specific user defined subflow.
5. Send the given payload (can be text, binary or both) to the server by fragmenting it equally across all available subflows (using the ad hoc scheduler).
6. Terminate the MPTCP connection.

4.2 Server Side

The server side of our methodology is made up of the *Snort* IDS, the *MPTCP server*, a log file generated by Snort and the *Log Analyzer* tool. Snort was described in Sect. 3, and the remaining components are discussed below.

MPTCP Server. A virtual remote machine with Ubuntu and the MPTCP kernel implementation [20] is used as the server. A simple off-the-shelf server (*http-server* [19]) is utilized to listen to incoming connections on port 80. The server accepts any incoming MPTCP connections on port 80 and receives the data. It is powerful enough to deal with a high number of simultaneously incoming requests.

Log Analyzer. Manually analyzing the log file generated by an IDS is a hectic task. Log Analyzer aims to automate this process. Snort writes its alerts to a CSV formatted log file in real time. Every value is at a fixed position on each line of the log file and can be extracted. The *Log Analyzer* reads the CSV file, parses and extracts the important features and then displays an output table similar to Table 1. We are thus able to see how many alerts per rule category exist in the log file.

Table 1. Output of the Log Analyzer tool.

Index	Category	Triggered alerts

5 Statistical Analysis of Snort Rules

This section presents the results of a statistical analysis conducted on the latest Snort rules. The motivation for this study was an investigation of how much of the Snort rule set might be affected by the cross-path data fragmentation attack using MPTCP. Thus only the rules relating to the TCP protocol were of interest. The results are based on Snort registered user rules v2.9 (snapshot 2970). The Rule Analyzer described in Sect. 4 is used to perform the analysis.

5.1 Results

Table 2 shows the results when the Rule Analyzer tool processes the Snort rules folder with all the relevant rule files in it. Table 3 shows the breakdown of TCP rules.

Table 2. Distribution of Snort rules v2.9 (excluding deleted rules).

Protocol	TCP	UDP	ICMP	IP	Total
Rules	18577 (84.17 %)	3134 (14.20 %)	156 (0.70 %)	203 (0.91 %)	22070

Table 3. Break down of TCP rules.

TCP	Rules with *content*	Rules with *offset*	Testable	Untestable
18577	18398 (99.03 %)	959 (5.16 %)	9857 (53.06 %)	8720 (46.93 %)

5.2 Trends

To investigate the evolution of Snort rules and any possible trends, the results obtained can be compared to older rule sets. However, due to licensing issues, it is not easy to acquire old Snort rules. An old study conducted on Snort rules v2.4 [17] is nevertheless relevant. In addition, two old rule set versions from October 2000 and June 2001 were found at [1]. The results of the old study on rules v2.4 and the breakdown of rules available at [1], performed by the Rule Analyzer tool that was developed, are shown in Table 4.

On the basis of these results, Fig. 4 shows a comparison chart. The following observations can be made from the comparison: (1) the total number of rules have increased significantly over the years (from 422 in October 2000 to 22070 today; (2) IP rules were non-existent at least until June 2001; (3) ICMP rules have increased but not considerably (from 47 to 156); (4) UDP rules have increased by a factor of 47 (from 66 to 3134); (5) TCP rules have increased rapidly by a factor of 60 (from 309 to 18577); and (6) TCP rules dominate other protocol rules over the years.

Table 4. Distribution of Snort rules v2.4, June'01 and Oct'00.

Rule Set	TCP	UDP	ICMP	IP	Total
v2.4	6494	356	132	39	7021
June 2001	404	86	55	0	545
October 2000	309	66	47	0	422

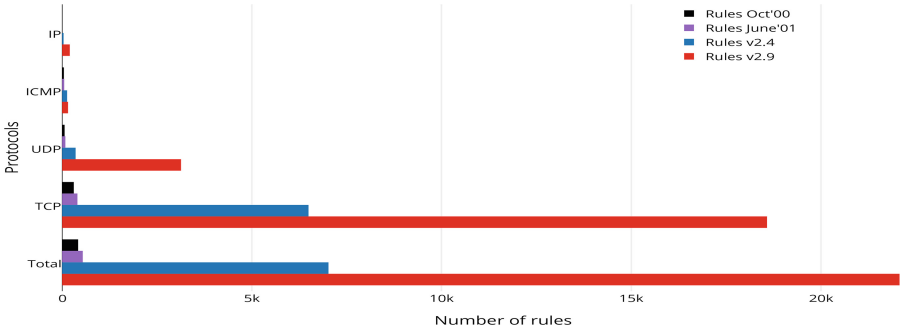

Fig. 4. Bar chart with a comparison of the number of rules in different rule sets.

6 Evaluation of Snort

This section will discuss the outcome of implementing the methodology discussed in Sect. 4. The overall operation is described next, and is followed by the results and a discussion.

6.1 Operation

The experimental methodology (see Fig. 3) discussed in Sect. 4 was put into operation in the following way. First, every rule category[4] file of Snort rules v2.9 is provided to the *Rule Analyzer* tool, which performs the analysis and classifies the rules into different protocols. Since UDP and ICMP rules will not be affected by any change at the transport layer, they are discarded. The remaining TCP rules are further classified into testable and untestable rules by the *Rule Analyzer* depending on whether the *Rule Parser* tool can create an accurate payload for them. Next, the testable TCP rules of every category are used as input to the *Rule Parser* tool to create a unique payload for each rule. These payloads are passed on to the *MPTCP tool*, which acts as a client. It establishes an MPTCP connection with the server, adds additional subflows to the connection and sends the payload across by equally fragmenting it among the available subflows. Every payload is tested using up to five subflows, starting from one subflow (which is equivalent to traditional TCP).

[4] According to the new Snort rule categories.

On the server side, a software-based middlebox Snort IDS runs with the same rules as were used to create the payloads. Every packet received on the server is logged and analyzed in detail by Snort before its being passed on to the Multipath TCP capable TCP/IP stack. Ideally, Snort should generate an alert for every rule due to the fact that all the incoming packets are specially crafted to trigger an alert. Snort logs its alerts in an alert file that is inspected by the *Log Analyzer* to extract the results discussed in the next subsection.

6.2 Results

Table 5 shows the test results for all Snort rule categories. For every category, an evaluation is conducted five times by changing the number of subflows and effectively increasing data fragmentation. The results with one subflow can be considered to be the baseline, since an MPTCP connection with one subflow is essentially a traditional TCP connection.

6.3 Discussion

The findings in Table 5 are of considerable significance. The results show that the Rule Parser tool has some inaccuracies. This leads to a lower number of triggered alerts than expected. Nevertheless, the number of alerts generated by Snort should be the same for any number of subflows, whether it be one subflow (equivalent to a traditional TCP session) or five subflows. This is not the case according to the results[5]. It can be observed from the results that, in general, the number of subflows in an MPTCP connection has an inversely proportional relationship with the detection capability of Snort. As the number of subflows increase (so does the data fragmentation), the detection of Snort suffers even more. With the fragmented data, Snort is only able to get partial matches of the signatures it is looking for in the packets. Snort is still able to detect some intrusions, even with five subflows. A deep look at those rules reveals a common characteristic. All such rules that still work search for a very small signature. Therefore, the MPTCP tool was not able to equally fragment the corresponding data stream of those rules using all available subflows.

The rationale behind this degraded detection capability of Snort is the lack of MPTCP awareness. Snort interprets all TCP subflows within an MPTCP connection as independent TCP connections. It has no awareness that multiple subflows could actually be components of the same MPTCP connection. Snort analyzes every TCP subflow in isolation and expects to see all the traffic required for matching a signature in that TCP session. That is not the case with an MPTCP connection of multiple subflows. The intrusion data are present in the overall MPTCP session (fragmented across subflows), and Snort can still not detect it. This behavior was referred to as *single-session bias* in a recent IETF draft [15]. These results confirm the concerns raised by [23]. The lack of MPTCP

[5] About one half of the Snort rule set is evaluated, but similar results are expected from the remaining rules.

Table 5. Results from all categories.

Index	Category	Subflows	Payloads sent	Triggered alerts
1	FILE	1	2064	2018
		2	2064	120
		3	2064	110
		4	2064	98
		5	2064	97
2	PROTOCOL	1	151	123
		2	151	9
		3	151	8
		4	151	8
		5	151	8
3	POLICY	1	364	340
		2	364	31
		3	364	28
		4	364	27
		5	364	25
4	SERVER	1	1753	1475
		2	1753	951
		3	1753	951
		4	1753	935
		5	1753	928
5	BROWSER	1	963	946
		2	963	8
		3	963	5
		4	963	4
		5	963	4
6	MALWARE	1	2539	1959
		2	2539	679
		3	2539	565
		4	2539	576
		5	2539	537
7	OS	1	288	271
		2	288	57
		3	288	52
		4	288	51
		5	288	50
8	INDICATOR	1	339	269
		2	339	70
		3	339	60
		4	339	57
		5	339	60
9	PUA	1	617	527
		2	617	449
		3	617	413
		4	617	351
		5	617	211
10	MISC.	1	779	680
		2	779	362
		3	779	156
		4	779	146
		5	779	154

awareness on the part of middleboxes performing traffic inspection is the primary reason why they are configured to force MPTCP into using traditional TCP instead. Essentially, these middleboxes currently see MPTCP traffic as attack traffic. If they allow it to pass, then the outcome is degraded functionality and degraded security.

7 Proposed Solution

We introduce the *MPTCP Linker*, a tool that captures and analyzes MPTCP packets to link sessions and correlate MPTCP subflows with the goal to mitigate the above discussed cross-path fragmentation attack. Using the MPTCP options, TCP flags and a few tricks, it is able to associate MPTCP subflows along with the data sent on them with the respective MPTCP connections (something Snort can not do). Figure 5 depicts the higher level concept behind the tool and the forthcoming subsections discuss the implementation and the validation of the tool.

7.1 Implementation

MPTCP Linker is implemented as a python script and is available at [2]. It sniffs the chosen network interface card for MPTCP packets and performs its processing using few commodity and open source modules. As output, it can generate TCP based pcap files. A separate pcap file is generated for each MPTCP

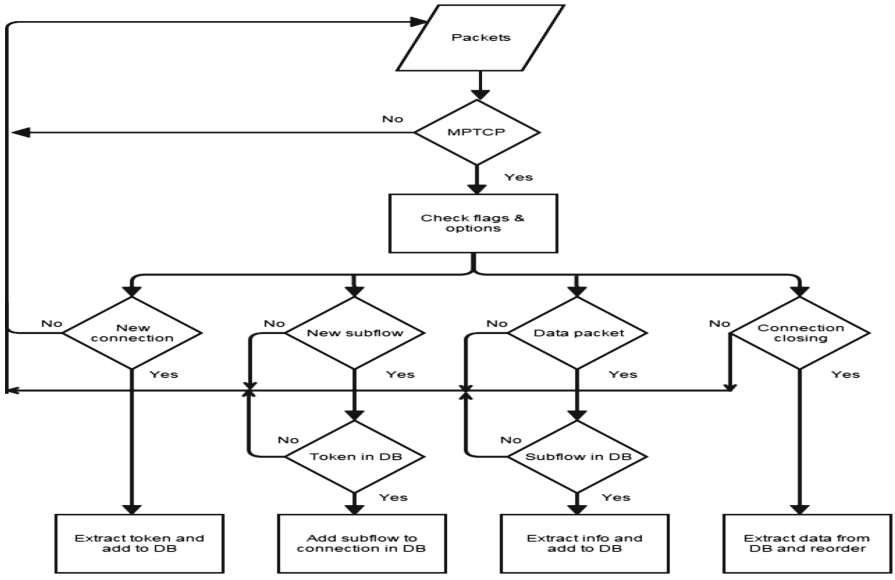

Fig. 5. Flow chart for operation of the MPTCP linker

Table 6. Validation of the MPTCP linker.

Category	Subflows	Payloads Sent	Alerts (before[6])	Alerts (after[7])
PROTOCOL	1	151	123	123
	2	151	9	123
	3	151	8	123
	4	151	8	123
	5	151	8	123

[6] Snort performed detection directly.
[7] Snort performs detection using packets generated by the MPTCP Linker.

session. Every pcap file contains in-order data from all subflows of that MPTCP session.

7.2 Validation

We validate the MPTCP Linker under the same attack traffic that was used for the evaluation of Snort in Sect. 6. Snort has an offline mode where it reads pcap files and detects intrusions from the packets. This offline mode is utilized for the validation. MPTCP Linker runs on the server side and performs its processing on all attack traffic to generate pcap files. The pcap files are then used by Snort in offline mode to perform intrusion detection. Table 6 shows the evaluation results from one category (due to space limitations) of Snort rules.

As can be seen in Table 6, the MPTCP Linker mitigates the earlier discussed cross-path fragmentation attack. The number of intrusions detected by Snort are consistent (123), irrespective of whether one subflow or up to five subflows are used to fragment the data stream. The MPTCP Linker correlates MPTCP subflows, links them to their respective MPTCP sessions and keeps track of the data on those subflows. Once it detects the termination of a session, it extracts data from all subflows of that session and reassembles them to recreate the original data stream in the correct order.

8 Outlook

The ultimate solution to this problem will be the evolution of the network security infrastructure to fully support MPTCP. However, there could be interim solutions that can be employed in the meantime to partially support the protocol and still ensure security. One such solution has been proposed and implemented in this work in the form of the MPTCP Linker [2]. There are also suggestions for developing TCP-MPTCP (and vice versa) proxies or a protocol converter [8]. Such solutions can help in the deployment of MPTCP as well as benefit network security.

9 Concluding Remarks

This paper investigated the reasons why existing network security middleboxes block MPTCP connections. With the help of a systematic experimental methodology, it was established that MPTCP could indeed be used by an attacker to degrade the functionality of existing network security middleboxes. To take a step towards adding MPTCP support and making security middleboxes MPTCP aware, a solution was also proposed and evaluated. The solution has been released under an open-access license for the benefit of the whole community. It also merits mentioning that only one of potentially many issues has been investigated in this work. Other security issues that arise with the advent of MPTCP, particularly the common scenario where only partial traffic passes through security middleboxes, also need to be explored and resolved.

Acknowledgments. The work was carried out in the High Quality Networked Services in a Mobile World (HITS) project, funded partly by the Knowledge Foundation of Sweden. The authors are grateful for the support provided by Catherine Pearce of Cisco.

References

1. Advanced Reference Archive of Current Heuristics for NIDS: Arachnids event signatures export for snort (2000–2001). http://www.autoshun.org/downloads/vision.conf, http://www.autoshun.org/downloads/vision18.conf
2. Afzal, Z.: MPTCP-Linker (2015). https://github.com/zafzal/MPTCP-Linker
3. Afzal, Z., Lindskog, S.: Automated testing of IDS rules. In: Proceedings of the 8th International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 1–2. IEEE, April 2015
4. Armitage, G., Williams, N., et al.: FreeBSD kernel patch to enable Multipath TCP (2014). http://caia.swin.edu.au/urp/newtcp/mptcp/tools.html
5. Bonaventure, O.: Apple seems to also believe in Multipath TCP (2013). http://perso.uclouvain.be/olivier.bonaventure/blog/html/2013/09/18/mptcp.html
6. Braun, M.B., Paasch, C., Gont, F., Bonaventure, O., Raiciu, C.: Analysis of MPTCP Residual Threats and Possible Fixes. Internet Draft draft-ietf-mptcp-attacks-02, IETF (2014). https://tools.ietf.org/id/draft-ietf-mptcp-attacks-02.txt
7. Detal, G.: MPTCP-enabled kernel for the nexus 5 (2014). https://github.com/gdetal/mptcp_nexus5
8. Detal, G., Paasch, C., Bonaventure, O.: Multipath in the middle (box). In: Proceedings of the 2013 Workshop on Hot Topics in Middleboxes and Network Function Virtualization, pp. 1–6. ACM (2013)
9. Ford, A., Raiciu, C., Handley, M., Bonaventure, O., et al.: TCP extensions for multipath operation with multiple addresses. Experimental RFC 6824, IETF (2013). https://tools.ietf.org/html/rfc6824
10. Giovanni, C.: Fun with packets: Designing a stick (2002). http://repo.hackerzvoice.net/depot_ouah/dos_ids.html
11. Han, H., Shakkottai, S., Hollot, C., Srikant, R., Towsley, D.: Multi-path TCP: a joint congestion control and routing scheme to exploit path diversity in the internet. IEEE/ACM Trans. Networking **14**(6), 1260–1271 (2006)

12. Honda, M., Nishida, Y., Raiciu, C., Greenhalgh, A., Handley, M., Tokuda, H.: Is it still possible to extend TCP? In: Proceedings of the 11th ACM SIGCOMM Internet Measurement Conference (IMC), pp. 181–194. ACM (2011)
13. Huitema, C.: Multi-homed TCP. Internet Draft draft-huitema-multi-homed-01, IETF (1995). https://tools.ietf.org/html/draft-huitema-multi-homed-01
14. Langley, A.: Probing the viability of TCP extensions (2008). http://www.imperialviolet.org/binary/ecntest.pdf
15. Lopez, E.: Multipath TCP middlebox behavior. Internet Draft draft-lopez-mptcp-middlebox-00, IETF (2014). https://tools.ietf.org/html/draft-lopez-mptcp-middlebox-00
16. Manev, P.: Rule2alert (2014). https://github.com/pevma/rule2alert
17. Münz, G., Weber, N., Carle, G.: Signature detection in sampled packets. In: Proceedings of the Workshop on Monitoring, Attack Detection and Mitigation (MonAM). IEEE (2007)
18. Mutz, D., Vigna, G., Kemmerer, R.: An experience developing an IDS stimulator for the black-box testing of network intrusion detection systems. In: Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC), pp. 374–383. IEEE (2003)
19. Nodejitsu: http-server (2014). https://github.com/nodeapps/http-server
20. Paasch, C., Barré, S., et al.: Multipath TCP implementation (v0.88) in the Linux kernel (2013). http://www.multipath-tcp.org
21. Patton, S., Yurcik, W., Doss, D.: An achilles heel in signature-based IDS: squealing false positives in SNORT. In: Proceedings of 4th International Symposium on Recent Advances in Intrusion Detection (RAID) (2001)
22. Pearce, C.: MPTCP roams free (by default!) (2014). http://labs.neohapsis.com/2014/10/20/mptcp-roams-free-by-default-os-x-yosemite/
23. Pearce, C., Thomas, P.: Multipath TCP: breaking today's networks with tomorrow's protocols. In: Black Hat USA, August 2014
24. Roesch, M., et al.: Snort: lightweight intrusion detection for networks. In: Proceedings of the 13th Conference on Systems Administration, pp. 229–238 (1999)
25. Stallman, R.: GNU General Public License, version 2 (1991)
26. Stewart, R.: Stream control transmission protocol. RFC 4960, IETF (2007). https://tools.ietf.org/html/rfc4960
27. The Snort Team: Snort official website. https://www.snort.org/
28. Thomas, P.: mptcp-abuse (2014). https://github.com/Neohapsis/mptcp-abuse
29. Wischik, D., Handley, M., Braun, M.B.: The resource pooling principle. ACM SIGCOMM Comput. Commun. Rev. **38**(5), 47–52 (2008)