Global Information Assurance Certification Paper

Practical approaches for MTCP Security

GIAC (GCIA) Gold Certification

Author: Josh Lewis, Joshua.d.Lewis@gmail.com

Advisor: Rob VandenBrink

Accepted: Date

Template Version September 2014

Abstract

Multi-path TCP (MPTCP) is an emerging IETF standard for providing connection resilience and bandwidth aggregation. MPTCP evolves the existing TCP protocol by allowing multiple TCP flows for a TCP session. This provides exciting new possibilities for mobile devices that can maintain TCP sessions as connection paths are added or dropped, and multi-homed servers that allow TCP sessions to take advantage of a mesh topology. However, current network security monitoring infrastructure solutions cannot appropriately inspect MPTCP connections, leaving significant intrusion detection and data loss blind spots. This paper will discuss practical approaches for MPTCP security.

1. Background

The goal of this research is to outline practical approaches to mitigate risks from Multipath TCP (MPTCP) using current security tooling. This research will discuss the MPTCP architecture, impact on security, use by an attacker, practical security approaches, and future research.

MPTCP builds on top of the TCP protocol. Readers that are unfamiliar, or that have not recently worked with the TCP are encouraged to brush up on connection setup, data transfer, connection teardown, and header options, which is outlined in section 10.1.

2. MPTCP architecture

2.1. MPTCP use cases

Multipath TCP (MPTCP) is a draft IETF standard that extends TCP and enables the simultaneous use of multiple IP addresses and ports (Ford, et al., 2011). MPTCP has two primary use cases that cater to mobile endpoints and datacenters. Mobile endpoints typically have built in cellular and 802.11 wireless network interfaces that have their own IP addresses and paths to the Internet. A traditional TCP session established over an 802.11 wireless network will be lost if the user roams outside of the access point coverage area. Likewise, a TCP connection established over a cellular network will not utilize the additional bandwidth if an 802.11 wireless network connection becomes available. MPTCP addresses these challenges by utilizing both interfaces to setup one or more independent TCP sessions, called sub-flows, which are presented to the application layer as one transparent connection. MPTCP will dynamically add or remove connection paths, enabling session resiliency and the use of the collective bandwidth from each connection path. Similarly, datacenters may have a mesh network with multiple paths between endpoints. A traditional TCP connection can only use one path at a time to exchange data. However, MPTCP sub-flows can be established over each path and used to aggregate the bandwidth from each connection.

Josh Lewis, Joshua.d.Lewis@gmail.com

2.2. MPTCP conceptual overview

MPTCP can be thought of as a transport layer shim that interfaces with the application layer and one or more TCP sessions (reference Figure 1). Applications are not required to be MPTCP aware, and can interact with the MPTCP shim without modification to the existing application code. The MPTCP shim receives data from the application layer, and segments the data across the different TCP sub-flows. Each TCP sub-flow receives data from the MPTCP shim and functions as an independent TCP session, which manages segmentation, encapsulation, and re-transmission. The MPTCP shim manages the setup of TCP sub-flows, segmentation of data received from the application layer, and reassembly of data received from the TCP sub-flows.

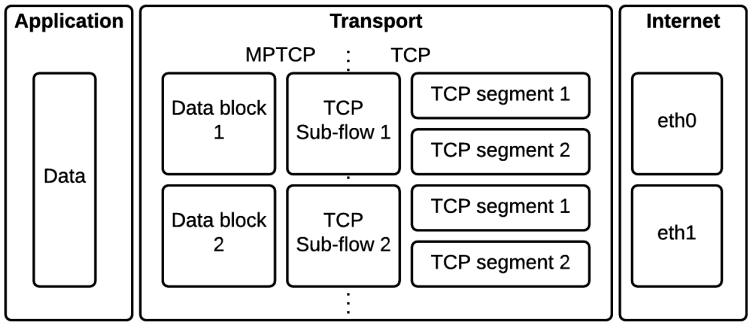

Figure 1 - MPTCP conceptual layers example, concept derived from (Ford, et al., 2011)

2.3. MPTCP three-way handshake and interface discovery

To setup a MPTCP session an originating host will initiate a TCP three-way handshake with the SYN flag set, an initial SYN sequence number, and a MPTCP option that 1) specifies that the host is MPTCP capable and 2) provides a token generated by the originating host to be used for authenticating sub-flows.

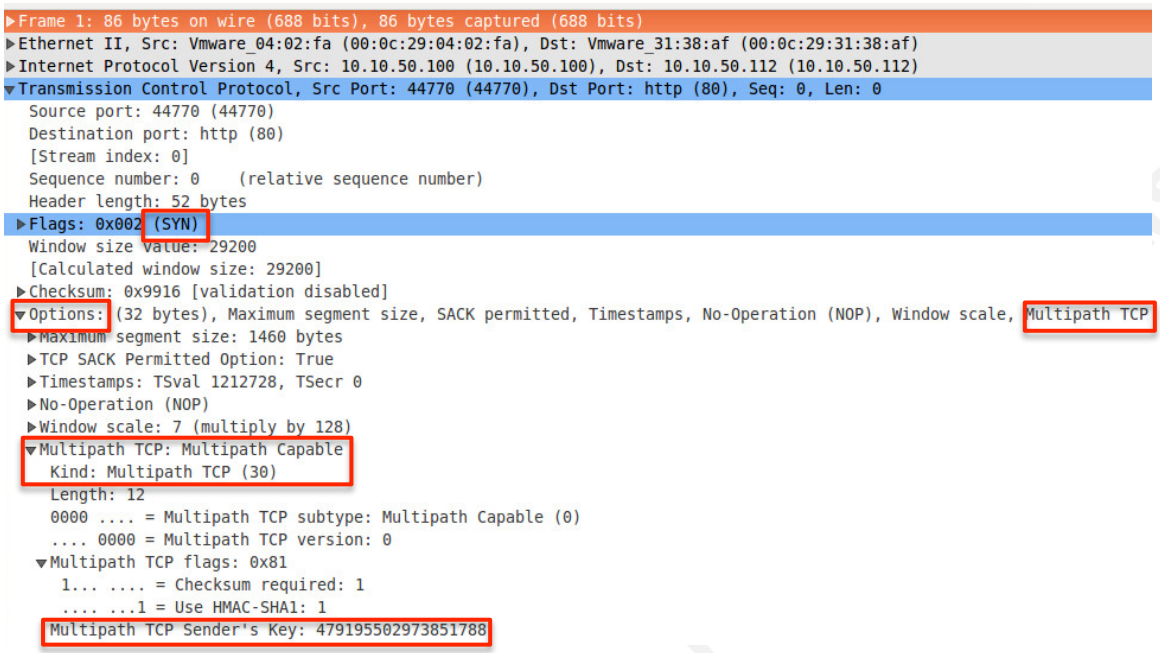

Figure 2 - MPTCP three-way handshake: SYN

If the remote host is listening on the corresponding port and also supports MPTCP, the remote host will reply with the SYN and ACK flag set, an initial SYN sequence number, the corresponding ACK sequence number, and a MPTCP option that 1) specifies that the host is MPTCP capable and 2) provides a token generated by the remote host that can be used for authenticating sub-flows.

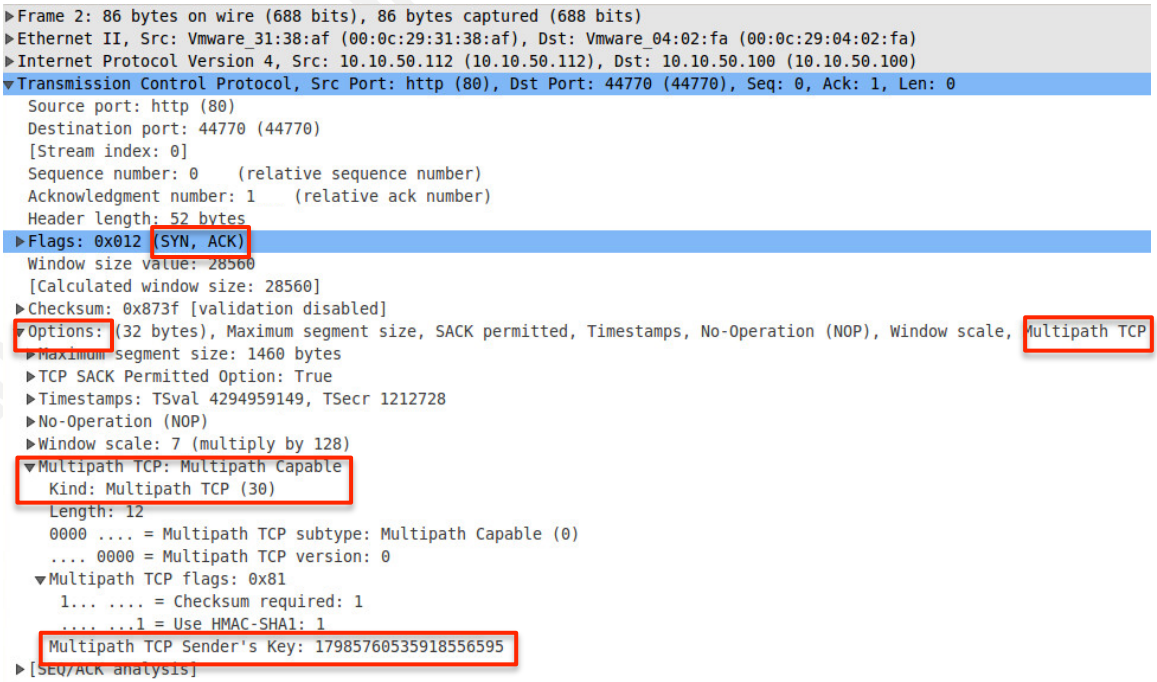

Figure 3 - MPTCP three-way handshake: SYN/ACK

Josh Lewis, Joshua.d.Lewis@gmail.com

If the remote host does not support MPTCP, the remote host reply will not include any MPTCP options, signaling to the originating host to default back to traditional TCP.

If the remote host replies with MPTCP capable, the originating host will respond with the ACK flag set, the corresponding ACK sequence number and an MPTCP option containing the originating and remote host tokens.

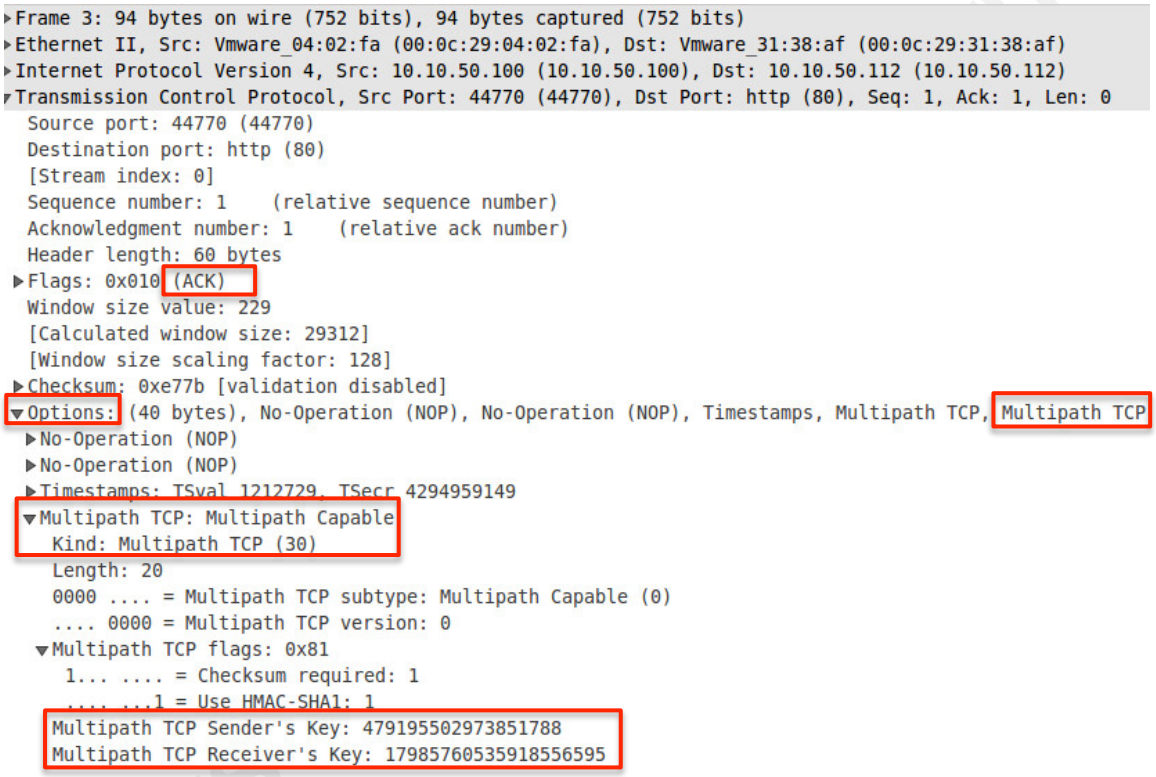

Figure 4 - MPTCP three-way handshake: ACK

Once the three-way handshake is completed, the originating host will set the ACK flag, the corresponding acknowledgment number, and an MPTCP option that contains 1) additional IP addresses on the originating host that can be used by the remote host and 2) the originating hosts data sequence signal initial sequence number. MPTCP uses an additional set of SYN and ACK sequence numbers called Data Sequence Signal (DSS). DSS SYN and ACK sequence numbers order the segments received from each of the sub-flows and allow data to be retransmitted when a sub-flow connection is lost. The remote host will reply with an ACK flag set, the corresponding ACK sequence number, and an MPTCP option that contains 1) information on additional IP addresses on the remote host

that can be used by the originating host and 2) the remote hosts DSS initial sequence number. Section paraphrased from: (Bonaventure, 2013).

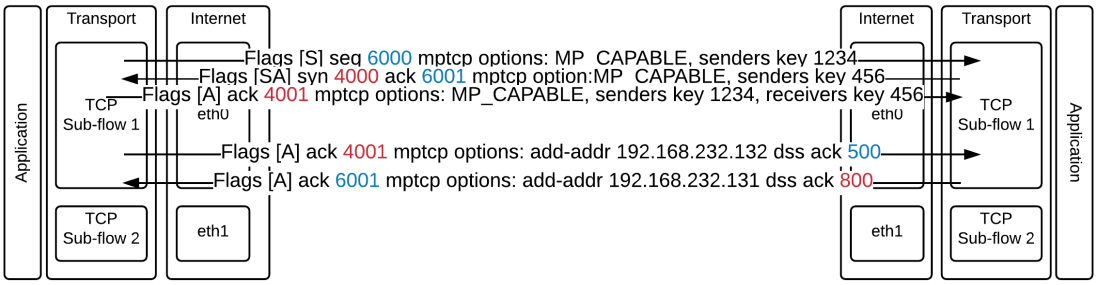

Figure 5 – Putting it all together: MPTCP three-way handshake with MPTCP option

2.4. MPTCP data transfer and sub-flow addition

Once the TCP three-way handshake for the first TCP sub-flow is complete, data can begin to be exchanged. The originating host will set the PUSH and ACK flags, the SYN and ACK sequence number corresponding to the three-way handshake, and a MPTCP DSS SYN and ACK sequence number corresponding to the originating and remote host initial DSS sequence numbers. The sub-flow TCP SYN and ACK sequence numbering aid in connection level segment sequencing and retransmission. MPTCP also used the DSS SYN and ACK sequence numbers for data level segmentation, sequencing and retransmission across the sub-flows. After the first payload is sent, MPTCP will setup additional TCP sub-flows. The originating host will initiate the TCP sub-flow three-way handshake to the IP address advertised by the remote host that was previously sent in the MPTCP option by setting the SYN flag, setting a new sub-flow specific initial SYN sequence number, and a MPTCP option that contains 1) MPTCP join request, and 2) a nonce value for the originating host. The remote host will reply with the SYN and ACK flags set, an initial SYN sequence number that is specific to the sub-flow, a corresponding ACK sequence number, and a MPTCP option that contains 1) a MTCP connection join request, 2) a nonce value for the remote host, and 3) a truncated HMAC of (the originating host nonce, the remote host key [note that the originating and remote host keys were exchanged during the initial TCP three-way handshake for sub-flow number one). The originating host will respond with the ACK flag set, a corresponding ACK sequence number, and an MPTCP option containing 1) MPTCP join request, and an

HMAC of (the remote host nonce, the originating host key). The second TCP sub-flow is now authenticated and can be utilized to send and receive data. Section paraphrased from: (Bonaventure, 2013).

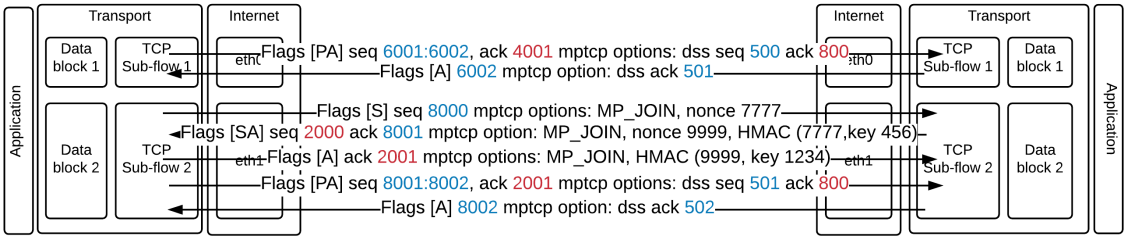

Figure 6 – Putting it all together: MPTCP data transfer and sub-flow addition

2.5. Graceful MPTCP connection termination

After the data exchange is complete, the MPTCP session can be gracefully or abruptly terminated. In a graceful termination, an originating host will set the ACK flags, a corresponding ACK sequence number, and an MPTCP option containing 1) the DSS SYN and ACK sequence numbers and 2) a DATA_FIN to indicate the data transfer is complete. The remote host will reply with an ACK flag set, and an MPTCP option containing 1) the corresponding DSS SYN and ACK sequence number and 2) a DATA_FIN to indicate the data transfer is complete. The originating host will then close the TCP sub-flows in a similar manner as a traditional TCP session. For each sub-flow, the originating host will send the FIN and ACK flags and the appropriate TCP SYN and ACK sequence numbers. The remote will reply with the FIN and ACK flags set and the appropriate TCP SYN and ACK sequence numbers. The originating host will set the ACK flag and send the appropriate ACK sequence number. Section paraphrased from: (Bonaventure, 2013).

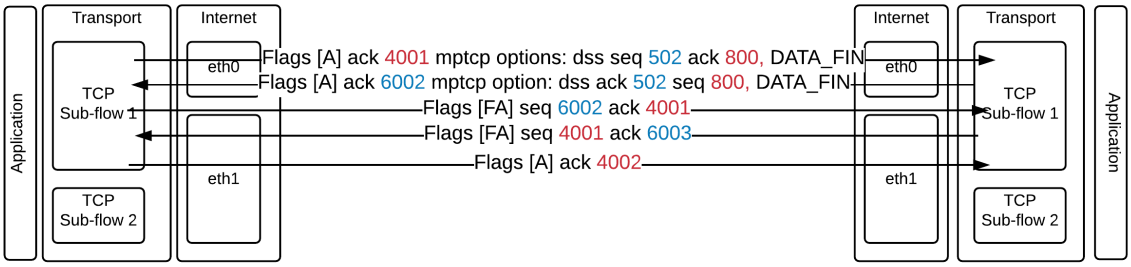

Figure 7 – Putting it all together: MPTCP graceful connection close

Josh Lewis, Joshua.d.Lewis@gmail.com

2.6. Aborted MPTCP connection termination

A MPTCP connection can also be abruptly aborted, similar to a TCP reset. Sending a TCP reset to a MPTCP sub-flow will result in the TCP connection being dropped, but the MPTCP connection will continue over the remaining sub-flows. The MPTCP layer must initiate the connection termination. The remote host will set the ACK flag, the appropriate ACK sequence number, and an MPTCP option that includes FAST_CLOSE. The originating host will respond by setting the RST flag and the appropriate SYN and ACK sequence numbers that correspond to the data sent by the originating host and the data sent by the remote host. Once the TCP sub-flow connection is reset is sent, no additional segments will be sent across this sub-flow. The originating host will then send a RST with the appropriate ACK sequence number to the remaining TCP sub-flows. Section paraphrased from: (Bonaventure, 2013).

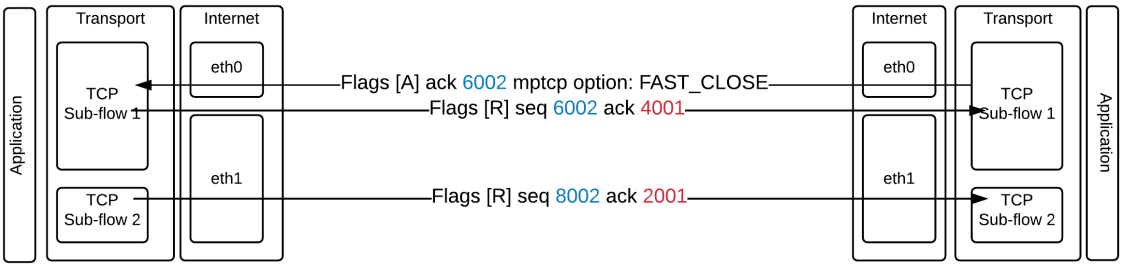

Figure 8 – Putting it all together: MPTCP aborted connection close

2.7. MPTCP options

MPTCP is designed to natively function on the majority of existing networks that support TCP. This is achieved by extending the TCP header through the use of TCP options. TCP options were designed to be an organic way to introduce new functionality without major changes to the protocol, such as the selective acknowledgement option. TCP options also have a lower chance of getting modified as they traverse through network filtering infrastructure (Bonaventure, 2013). Therefor MPTCP uses a single variable length TCP option to pass connection connection setup and maintenance information. MPTCP is identified by a TCP option number 30 (1e in hex), a four bit length field to specify the size of the MPTCP option data, a four bit sub-type field that identifies the MPTCP action to perform, and variable length sub-type data field.

Value	Symbol	Name
0x0	MP_CAPABLE	Multipath Capable
0x1	MP_JOIN	Join Connection
0x2	DSS	Data Sequence Signal (data ack and data sequence mapping)
0x3	ADD_ADDR	Add address
0x4	REMOVE_ADDR	Remove address
0x5	MP_PRIO	Change Sub-flow Priority
0x6	MP_FAIL	Fallback
0x7	MP_FASTCLOSE	Fast Close

Table 1 - MPTCP sub-type options (Cisco, et al., 2013)

The goal of this section is to provide a brief background on the architecture of MPTCP. RFC 6182 and RFC 6824 further discuss additional architectural details such as design considerations, congestion control, path priority, error handling, interaction with network filtering infrastructure, and the security of the protocol itself. Additionally, RFC 6181 discusses a threat analysis of the MPTCP protocol. The subsequent sections in this research will focus on why MPTCP is import for security and the practical approaches for MPTCP security.

3. MPTCP impact on information security

MPTCP provides significant connection resiliency and bandwidth improvements for mobile devices and meshed networked computing infrastructure in a datacenter. However, MPTCP presents two primary challenges for existing Intrusion Detection System (IDS), Intrusion Prevention System (IPS), and Data Loss Prevention (DLP) technologies. The first challenge is that MPTCP sub-flows may utilize side-channel cellular connections that enterprises may not be able to monitor. The second challenge is that network security infrastructure may not properly re-assemble data across sub-flows. These challenges enable attackers to evade IDS/IPS and DLP technologies. The

screenshot below showcases the ability to evade non-MPTCP aware signature detection technologies through the use of a netcat connection between two MPTCP capable hosts. The reassembled TCP conversation on eth0 captured the initial "evil" payload, while the reassembled TCP conversation on eth1 captured the "stuff" payload. An existing IDS/IPS or DLP technology with a signature for "evil stuff" would not alert on this traffic.

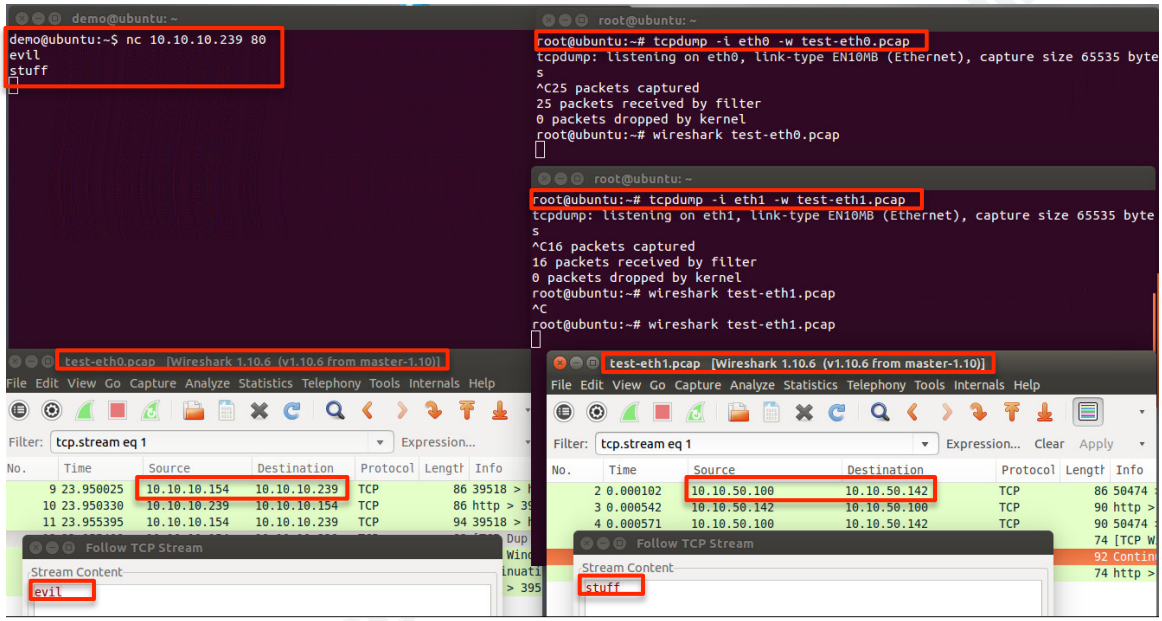

Figure 9 - MPTCP evasion example

Beyond enterprise network security evasion, MPTCP is also being explored to provide positive security benefits such as increasing anonymity and performance for the TOR network (Rochet, Pereira, & Bonaventure, 2015) or increasing connection privacy by utilizing cypher block chaining encryption across TCP sub-flows (Pearce C. , BSides Knoxville 2015 - Multipath TCP - Breaking Today's Networks with Tomorrow's Protocols, 2015). The scope of this paper does not address MPTCP use cases that can be used for a security benefit.

4. MPTCP market landscape

MPTCP is experiencing increasing adoption, however MPTCP is not broadly installed by default. Apple introduced MPTCP support in iOS 7 (Apple Inc., 2015) and OS X Yosemite (Pearce K. , 2014) however, MPTCP is not enabled for all TCP connections by default. Additionally, MPTCP extensions are available for some *nix and Android distributions (Barre & Paasch, n.d.). The table below summarizes common operating systems were tested for MPTCP support as September, 2015.

Table 2 **Operating System**	**MPTCP installed by default***	**MPTCP enabled by default (all connections)***	**MPTCP enabled by default (selective connections)***	**MPTCP extension available**
iOS 7-8	Yes	No	Yes, Apple Siri	N/A
OS X Yosemite	Yes	No	Unknown	N/A
Windows 7	No	No	No	No
Windows 8	No	No	No	No
Windows 10	No	No	No	No
Ubuntu 14.04	No	No	No	Yes
Ubuntu 15.04	No	No	No	Yes
Android 4.1-4.4.2	No	No	No	Yes

Table 3- MPTCP supporting operating systems as of September 2015

*Based on operating system testing as of September 2015.

Despite the fact that support and implementation of MPTCP is experiencing increasing adoption, few network security vendors are able to detect, filter or reassemble MPTCP traffic. The table below reflects MPTCP support and default configuration derived from publically accessible vendor documentation available on the internet as of September, 2015.

Security Product	Allow MPTCP by Default*	Reassemble MPTCP Sub-flows*	Capability to block MPTCP*	Revert MPTCP to TCP*	Reference
Cisco ASA	No	No	Yes	Yes	(Cisco, 2013)
Cisco ASA Next-Gen	Yes	No	Yes	Yes	(Cisco, 2013)
Cisco IPS	No	No	Yes	No	(Cisco, 2013)
Snort IDS	N/A	No	N/A	N/A	(Cisco, 2014)
Palo Alto Firewall	Yes	No	No	No	(harshanataraja n, 2015)
F5 BIG-IP	No	No	Yes	Yes	(F5, 2015)
Blue Coat	Could not find publically accessible documentation acknowledging support				
Checkpoint IPS	Unknown	No	Yes	Yes	(Check Point, 2014)
HP TippingPoint	Could not find publically accessible documentation acknowledging support				

Table 4 - MPTCP vendor landscape, September 2015

*Based on publically accessible vendor documentation available on the internet as of September, 2015.

5. MPTCP from an attackers perspective

Based on the ability to evade network security filtering infrastructure and the increasing prevalence of MPTCP, attackers will likely start utilizing MPTCP. From an attacker's perspective the primary abuse cases are to utilize MPTCP to evade: detection for the delivery of an exploit payload, command and control traffic, and data exfiltration. The instantiation of these abuse cases can be realized through MPTCP the setup and configuration of MPTCP on the target and attackers endpoints.

Josh Lewis, Joshua.d.Lewis@gmail.com

5.1. MPTCP fast flux

The optimal configuration for an attacker's infrastructure would be to leverage two or more publically addressable MPTCP network interfaces. Increasing the number of publically addressable MPTCP network interfaces will further fragment the application data across MPTCP sub-flows. An attacker may prefer to utilize a cloud based Infrastructure as a Service (IaaS) provider that can easily add several publically addressable MPTCP network interfaces. Since MPTCP can dynamically add and remove IP addresses during a session, an attacker may develop a capability that allows them to continuously add and release the IaaS public IP addresses throughout the duration of the MPTCP session. This concept is similar to DNS fast flux, but instantiated at a TCP session level.

5.2. MPTCP evasion with one network interface

After an attacker has setup their infrastructure, they can perform a SYN scan to determine if the target host supports MPTCP. If MPTCP is supported and not blocked by the network filtering infrastructure, the attacker can setup a TCP session with the listening service. Once the initial TCP session has been established, the target host will automatically setup sub-flows to each of the attackers MPTCP interfaces, even if the target host only has one network interface. If the target network filtering infrastructure is not MPTCP aware, the network filtering infrastructure will see independent, unrelated TCP sessions from one or more target host IP addresses to two or more attacker IP addresses. The data from each of these TCP sessions will be independently reassembled.

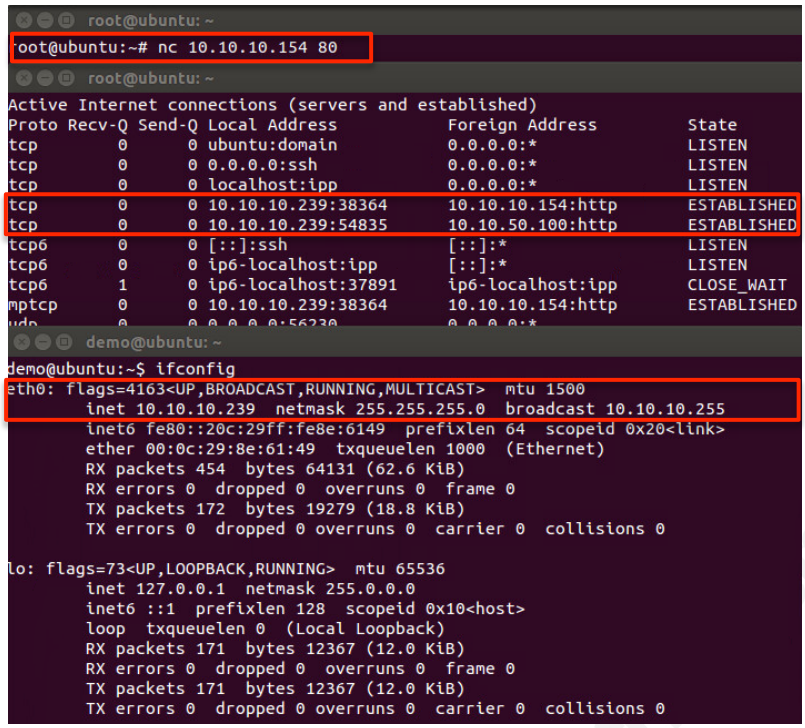

Figure 10 - Client with one MPTCP interface communicating with a server that has multiple MPTCP interfaces

In addition to using hosts that are have MPTCP built in or installed, an attacker can also install MPTCP extensions. As discussed in the previous section, MPTCP extension are available for *nix and Android devices. An attacker must download and install the extension, modify the boot loader to boot to the kernel with the MPTCP, and reboot the device (reference appendix section 10.1 for an example of setting up MPTCP). As long as the attacker has at least two publically addressable MPTCP network interfaces, application data will be split across sub-flows even if the target host only has one network interface. While this attack scenario does seem to require a bit more effort, it is still possible for an attacker to accomplish.

6. Practical approaches for MPTCP security

Based on the increasing prevalence of MPTCP, limited support in existing network security products, and how attackers will likely begin to utilize MPTCP, a MPTCP strategy should be employed to limit the risk of MPTCP abuse cases. In the near term (0-

12 months), MPTCP should be blocked for critical endpoints until network security vendors can appropriately reassemble and inspect this traffic. The near term strategy focuses on scanning the network to identify MPTCP capable endpoints, blocking MPTCP traffic at the host level, and monitoring for the use of MPTCP at the network level. Additionally, enterprises should issue support requests to network security vendors to provide the ability to reassemble MPTCP sub-flows traversing the same path and to provide the ability to terminate a malicious MPTCP connection.

6.1. Scanning the network for MPTCP capable endpoints

The first step to limit the abuse of MPTCP is to identify endpoints that are MPTCP capable. Patrick Thomas developed a MPTCP scanner that is built in python and utilizes Scapy to send a SYN to a listening port, with a TCP option 30 (MPCP), and sub-type 0 (MP_CAPABLE) [Reference Table 1 in section 2.7 to review TCP option 30 sub-types]. A MPTCP capable client can be identified by a response that has a SYN/ACK, TCP option 30, and sub-type 0. The script also performs a second check by sending a SYN to a listening port, with a TCP option 30, sub-type 1 (MP_JOIN). If the host is MPTCP capable it should respond with a RST, since a previous MPTCP session was not established (Thomas, 2014). Reference section 10.4 for an example of the setup and use of this MPTCP scanner. In large networks, consider scanning critical infrastructure first. Depending on the environment, critical infrastructure may not have a legitimate use case to utilize MPTCP.

6.2. Blocking MPTCP traffic at the host level

Based on the current network security infrastructure inability to appropriately reassemble and inspect MPTCP sub-flows, MPTCP should be blocked for critical endpoints to reduce the possibility of an IDS/IPS or DLP evasion. MPTCP traffic can be blocked at the host by disabling the MPTCP kernel parameter and by setting an MPTCP iptables rule. Depending on the host configuration and environment constraints, either or both of these methods can be utilized to effectively block MPTCP traffic at the host level.

The first method of blocking MPTCP at the host level is to configure the MPTCP kernel parameter. Modifying the MPTCP "enable" kernel parameter will turn off MPTCP for all TCP connections. Identify the MPTCP "enable" kernel parameter by running the command in Figure 11 (example output shown in Figure 12 and Figure 13). After the kernel parameter is identified set the value to zero (disabled) by running the two commands shown in Figure 14. The first command sets the MPTCP enabled parameter to zero in the sysctl.conf file, which will enable the configuration to persist across host restarts. The second command writes to the running MPTCP kernel parameter. These configuration parameters could be deployed through a script or mobile device endpoint agent. This method of controlling MPTCP is preferred based on ability to directly modify the MPTCP configuration, without TCP retransmission delays (discussed in the next section).

```
root@ubuntu:~# sysctl –a | grep mptcp
```

Figure 11 - MPTCP kernel parameters

```
root@ubuntu:~# sysctl -a | grep mptcp
kernel.osrelease = 3.14.0-89-mptcp
net.mptcp.mptcp_checksum = 1
net.mptcp.mptcp_debug = 0
net.mptcp.mptcp_enabled = 1
net.mptcp.mptcp_path_manager = fullmesh
net.mptcp.mptcp_scheduler = default
net.mptcp.mptcp_syn_retries = 3
root@ubuntu:~#
```

Figure 12 - Linux MPTCP kernel V0.89 enable kernel parameter

```
Laptop:~ root# sysctl -a | grep mptcp
net.inet.mptcp.enable: 1
net.inet.mptcp.debug: 0
net.inet.mptcp.mptcp_cap_retr: 2
net.inet.mptcp.dss_csum: 0
net.inet.mptcp.fail: 1
net.inet.mptcp.keepalive: 840
net.inet.mptcp.mpprio: 1
net.inet.mptcp.remaddr: 1
net.inet.mptcp.fastjoin: 1
net.inet.mptcp.zerortt_fastjoin: 0
net.inet.mptcp.rwnotify: 0
net.inet.mptcp.verbose: 0
net.inet.mptcp.pcbcount: 0
net.inet.mptcp.sk_lim: 16
net.inet.mptcp.delayed: 0
net.inet.mptcp.rto_spikethresh: 3000
net.inet.mptcp.force_64bit_dsn: 0
net.inet.mptcp.rto: 3
net.inet.mptcp.nrto: 3
net.inet.mptcp.tw: 60
Laptop:~ root#
```

Figure 13 - OS X Yosemite MPTCP enable kernel parameter

```
root@ubuntu:~# echo 'net.mptcp.mptcp_enabled=0' >> /etc/sysctl.conf
root@ubuntu:~#sysctl –w net.mptcp.mptcp_enabled=0
```

Figure 14 - Set MPTCP kernel parameter to disabled

The second method of blocking MPTCP at the host level is to utilize an iptables firewall rules. Figure 15 illustrates an iptables rule to block new, inbound TCP (-p TCP) traffic

with a TCP option 30 (MPTCP). A remote host that is MPTCP capable will send a SYN with a TCP option 30, subtype of 0x0 (MP_CAPABLE) to a local host on an open port. The local host iptables rule in Figure 15 will block this connection. The remote host will retransmit the TCP SYN with MPTCP capable packet based on the number of retries set in the kernel parameters (Linux MPTCP Kernel V 0.89 defaults to 3 retries). Each retry causes the remote host to wait prior to sending the next retry. After the MPTCP retry limit has exceeded, the remote host will send a SYN without an MPTCP option, resulting in the local host responding with a SYN/ACK. Although this method is effective in preventing an inbound MPTCP connection, it will delay the TCP connection. A host that defaults to three MPTCP retries will experience a ~15 second delay to complete the three-way handshake (reference Figure 17). This method of controlling MPTCP connections may not be appropriate for some use cases (e.g. web servers).

```
demo@ubuntu:~$ sudo iptables -A INPUT -p tcp --tcp-option 30 -m state --state
NEW -j DROP
```

Figure 15 - Block inbound MPTCP connections

Figure 16 illustrates an iptables rule to block new, outbound TCP (-p TCP) traffic with a TCP option 30 (MPTCP). This iptables rule will prevent the host from establishing a MPTCP session. However, some MPTCP implementations may default to initiating all TCP connections with MPTCP, causing TCP retries for each new TCP connection (reference Figure 17).

```
demo@ubuntu:~$ sudo iptables -A OUTPUT -p tcp --tcp-option 30 -m state --state
NEW -j DROP
```

Figure 16 - Block outbound MPTCP connections

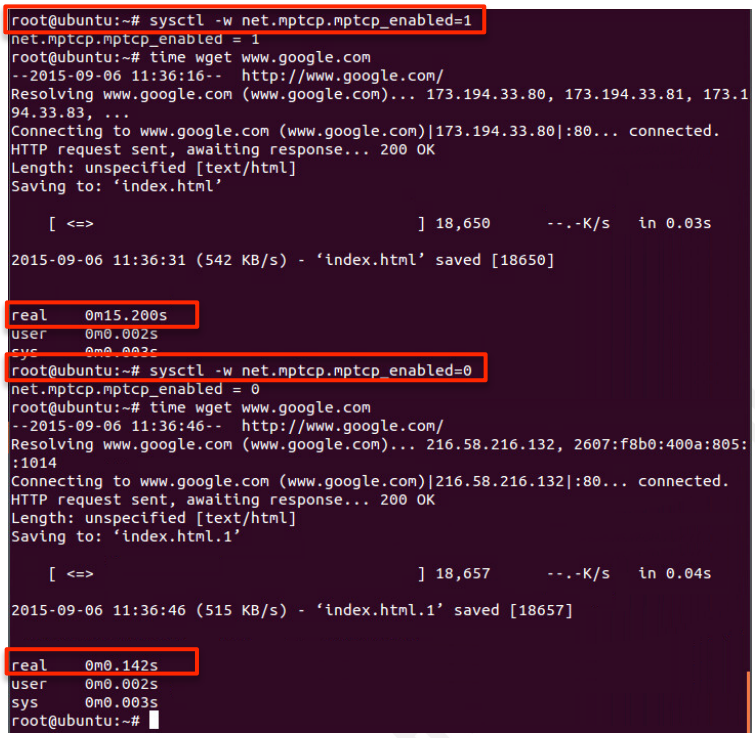

Figure 17 - MPTCP retransmission delay with iptables rules blocking TCP Option 30

6.3. Blocking MPTCP at the network level

MPTCP can be blocked at the network level in a similar manner to the host level iptables rules outlined above. Building and deploying these rules will vary depending on the network security vendor. However, three additional factors should be considered prior to implementing network level blocking of MPTCP traffic. First, some network security products do not allow granular blocking of TCP options (reference Table 4). Second, the scope of the network rules should be considered. Many user tier endpoints or IoT devices may have legitimate use cases for MPTCP. The initial network level blocking of MPTCP should focus on blocking MPTCP traffic to critical endpoints or endpoints that do not have a legitimate use case for MPTCP. Finally, endpoints that have side-channel connections (e.g. cellular interfaces) may initiate a connection over the side-channel. Blocking MPTCP on the enterprise interface may result in all traffic being pushed to the side-channel interface. Therefore, network level blocking of MPTCP should only be utilized if 1) MPTCP cannot be controlled at the host level due to a limited ability to control host configuration, 2) layer three filtering capabilities are placed directly in front of critical infrastructure where MPTCP filtering can be limited to these

endpoints, 3) critical infrastructure does not have multiple connection paths (e.g. side-channel connections), and 4) critical infrastructure use cases will not be effected by TCP retransmission delays.

6.4. Monitoring MPTCP traffic

After the critical infrastructure or the broader enterprise has been scanned to identify network stacks that support MPTCP, monitoring should be enabled for networks that should not be running MPTCP. However, poorly segmented networks will make monitoring difficult, since the scope of what is appropriate and inappropriate will be challenging to define. At the time that this research was conducted, some network security vendors were unable to log MPTCP traffic (reference Table 4). This research will highlight two options to monitor MPTCP traffic that are vendor agnostic.

The first MPTCP monitoring option (Figure 18) utilizes a tcpdump Berkeley Packet Filter (BPF) bit-mask. The goal of this filter is to identify an established session (SYN and ACK flags set [bit-mask 0x12] in TCP offset 13) and the TCP option 30 (30 decimal = 1e in hex) at offset 40.

```
demo@ubuntu:~$ tcpdump –i eth0 ‘tcp[13] & 0x12 = 0x12’ and tcp[40] = 0x1e
```

Figure 18 - Capture MPTCP established connections

The second MPTCP monitoring option utilizes a host iptables rule. Iptables is able to natively trigger actions by TCP option number. The command in Figure 19 appends (-A) a rule to the OUTPUT chain, to monitor the TCP protocol (-p), for option 30 (MPTCP), for a connection that is established (TCP three-way handshake completed), limiting the number of connections logged to five connections per minute (–m limit --limit 5/min), with a burst of five connections before the limit kicks in (--limit-burst 5), adding a unique string to the log entry “IPTBLSDROP: MTCP OUT ESTAB “, and logging the IP/TCP options. The command in Figure 20 is similar to the previous command, except it creates a log for incoming (INPUT) MPTCP connections. Reference section 10.3 for a sample iptables script. After the iptables rules are setup, rsyslog can be configured to send the

MPTCP events to a log file. The command in Figure 21 creates a configuration file that looks for the unique string created in the iptables rules and outputs the results to an mptcp.log file. Note that rsyslog monitors the /etc/rsyslog.d/ directory for any file name *.conf.

```
demo@ubuntu:~$ sudo iptables –A OUTPUT –p tcp --tcp-option 30 –m state --state ESTABLISHED –m limit --limit 5/min --limit-burst 5 –j LOG --log-prefix 'IPTBLSDROP: MTCP OUT ESTAB ' --log-ip-options --log-tcp-options --log-tcp-sequence
```

Figure 19 - Log outbound MPTCP established connections

```
demo@ubuntu:~$ sudo iptables –A INPUT –p tcp --tcp-option 30 –m state --state NEW –m limit --limit 5/min --limit-burst 5 –j LOG --log-prefix 'IPTBLSDROP: MTCP IN ATTEMPT ' --log-ip-options --log-tcp-options --log-tcp-sequence
```

Figure 20 - Log source hosts trying to establish a MPTCP connection

```
root@ubuntu:~# echo ':msg,contains,"IPTBLSDROP: MTCP" /var/log/mptcp.log' > /etc/rsyslog.d/mptcp.conf
```

Figure 21 - Configure rsyslog to extract iptables MPTCP events to a log file

6.4.1. Use of a proxy

Beyond blocking or logging TCP option 30 with a network firewall, a proxy can also be utilized to block or monitor the use of MPTCP traffic. Using a proxy to connect to external endpoints, a local host will setup a connection directly with the proxy. The proxy will setup an independent connection with the desired remote endpoint and pass traffic between the two established connections. Since the proxy performs the three-way handshake with the remote endpoint, the proxy has the capability to negotiate the supported TCP options. Depending on the vendor configuration, the proxy can forward the MPTCP option unchanged, strip the MPTCP option, or ignore the MPTCP option and negotiate TCP options based on the supported TCP options of the proxy. If the proxy strips or does not initiate the TCP three-way handshake with the TCP option 30, the

MPTCP capable remote endpoint will reply without the TCP option 30, since it perceives that the originating host does not support MPTCP. When the originating local host receives the connection from the proxy, the SYN/ACK reply will not contain the TCP option 30 and the local originating host will fail back to regular TCP. Similarly, if the proxy drops a sub-flow with the MP_JOIN, the sub-flow will not be utilized.

6.5. Network security vendor feature request

Scanning for MPTCP capable endpoints, blocking MPTCP connections to critical nodes, and monitoring network traffic, is a good starting points to manage the risk from MPTCP abuse cases. However, these initial steps should be paired with feature requests to network security vendors to provide the ability to reassemble MPTCP sub-flows that traverse a single path and for the capability to control a malicious MPTCP connection.

6.5.1. Reassemble MPTCP sub-flows that traverse a single path

The first feature request that should be submitted to network security vendors is for the capability to reassemble multiple TCP sub-flows that traverse the same path. As previously discussed in section 5, a host with one MPTCP capable interface can establish multiple sub-flows to a remote endpoint that maintains multiple MPTCP capable interfaces. An attacker can easily use this configuration to evade current network IDS/IPS and DLP infrastructure, even though all MPTCP traffic traverses the same enterprise network path. This capability should be a quick win for network security vendors. However, a long term solution also needs to be developed to inspect MPTCP traffic that utilizes a side-channel (non-enterprise network) connection.

6.5.2. Controlling a malicious MPTCP connection

The second feature request that should be submitted to network security vendors is for the capability to terminate an established MPTCP connection. An established MPTCP connection that has one or more sub-flows that traverse the enterprise network and one or more sub-flows that traverse a side-channel network will be difficult to terminate unless the enterprise controls the endpoint or the side-channel network. Using this example, an enterprise network security device that blocks the IP address or performs a TCP RESET will terminate the sub-flows traversing the enterprise network, resulting in all traffic

being pushed to the side-channel connections. At a minimum, network security vendors should be able to perform a MPTCP FAST_CLOSE to force the reset of all TCP sub-flows. Additionally, network security vendors may also provide the capability to tar pit a malicious MPTCP connection by dynamically updating the TCP window field to zero [similar to LaBrea Version 2 (The SANS Institute, 2015)]. The TCP window field specifies the amount of bytes that can be buffered on the receiving host, and indicates to the sending host how many bytes can be sent prior to an acknowledgement. MPTCP utilizes a shared window across all TCP sub-flows, permitting a LaBrea Version 2 like tar pit.

7. Future research

Based on the exploration of MPTCP in this paper, three areas were identified for further research. The additional research areas include building a more robust MPTCP scanner, determining how to utilize and disable MPTCP in OS X or iOS, and exploring the use of MPTCP DSS sequence numbers to retransmit payloads sent over side-channel connections.

7.1.1. Build an enhanced MPTCP scanner

The first area for future research focuses on building an enhanced MPTCP scanner. At the time that this research was conducted, there was only one proof of concept MPTCP scanner available for finding MPTCP capable end points. This scanner was built using Python and Scapy. Additional scanners should be explored that provide more robust functionality, such as the capability for multi-threading scanning.

7.1.2. Utilize MPTCP is OS X and iOS

The second area for future research focuses on utilizing MPTCP for OS X and iOS. At the time that this research was conducted, OS X Yosemite and iOS 7+ had an implementation of MPTCP installed, but was not used by default for all TCP connections. Enabling MPTCP on these platforms will allow the previously outlined MPTCP abuse cases to be realized on these platforms. Once MPTCP can be leveraged,

attackers can begin to use this for data exfiltration or obfuscation. This research should also discuss how to properly disable MPTCP if desired by enterprise security.

7.1.3. Retransmit MPTCP payloads sent through side-channels

The third area for future researches focuses on utilizing MPTCP DSS SYN sequence numbers to retransmit data sent over side-channel connections. A firewall, IDS/IPS, proxy or DLP solution that sees DSS SYN sequence number one and DSS SYN sequence number three traverse through the inspection engine, may be interested in issuing a retransmission for DSS SYN sequence number two, in order to fully reassemble the payload. At the TCP layer acknowledged data cannot be re-transmitted. Per the RFC 6824, data with the same sequence number can be retransmitted on a different sub-flow. Additionally, "An MPTCP sender MUST NOT free data from the send buffer until it has been acknowledged by both a data ACK received on any sub-flow and at the sub-flow level by all sub-flows on which the data was sent" (Cisco, et al., 2013). MPTCP should be tested to determine if a retransmission request can be reliably reissued over the enterprise path which is potentially faster than the side-channel path, or if the continuous retransmission of payload data sent over the side-channel connection will cause MPTCP to determine that the side-channel is unreliable and close the side-channel connection. The retransmission of DSS sequence numbers should be explored to aid in non-enterprise sub-flow visibility and inspection.

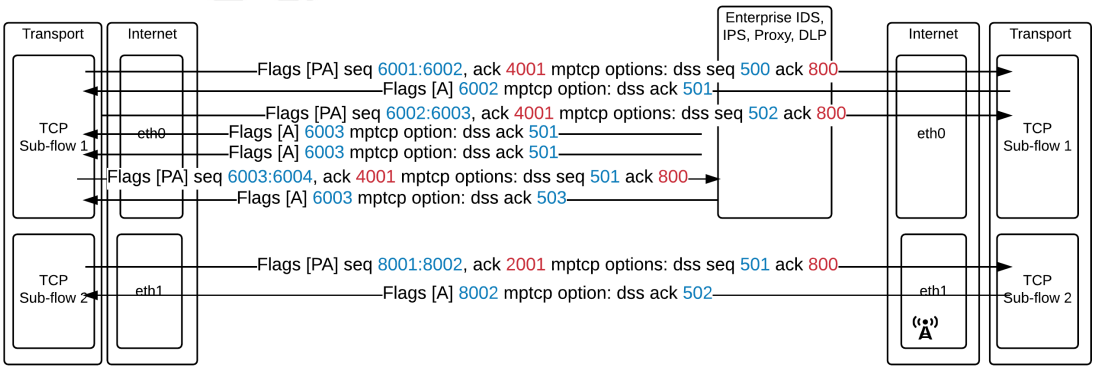

Figure 22 - Re-transmitting DSS sequence numbers for reassembly visibility

8. Conclusion

MPTCP is an exciting evolution of TCP that has performance and reliability benefits for user endpoints and datacenter infrastructure. MPTCP also presents some exciting opportunities to enhance security. The current adoption of MPTCP is increasing, with OS X Yosemite and iOS 7+ shipping with MPTCP installed and enabled in the kernel, as well as the availability *nix and Android extensions. However, current network security infrastructure lacks the ability to reassemble MPTCP sub-flows, resulting in possible IDS/IPS or DLP evasion abuse cases. Based on the lack of the current network security vendors ability to properly reassemble MPTCP sub-flows, MPTCP should be tightly controlled for critical endpoints by the identification of hosts with MPTCP enabled, blocking of MPTCP at the host level, and monitoring for MPTCP traffic at the network level. Additionally, enterprises should submit feature request to network security vendors for the ability to reassemble MPTCP sub-flows that traverse a single path and for the capability to control a malicious MPTCP connection.

9. References

Apple Inc. (2015, 4 1). *iOS: Multipath TCP Support in iOS 7*. Retrieved from support.apple.com: https://support.apple.com/en-us/HT201373

Barre, S., & Paasch, C. (n.d.). *MultiPath TCP - Linux Kernel Implementation*. Retrieved from multipath-tcp.org: http://multipath-tcp.org

Bonaventure, O. (2013, 03). *Decoupling TCP from IP with Multipath TCP.* Retrieved from multipath-tcp.org: http://multipath-tcp.org/data/MultipathTCP-netsys.pdf

Check Point. (2014, 9 1). *Check Point Advisories*. Retrieved from checkpoint.com: http://www.checkpoint.com/defense/advisories/public/2014/cpai-31-aug.html

Cisco. (2013, 9 17). *MPTCP and Product Support Overview.* Retrieved from Cisco.com: http://www.cisco.com/c/en/us/support/docs/ip/transmission-control-protocol-tcp/116519-technote-mptcp-00.html

Josh Lewis, Joshua.d.Lewis@gmail.com

Cisco. (2014). *SNORT Users Manual 2.9.7*. Retrieved from manual.snort.org: manual.snort.org

Cisco, Raiciu, C., Ford, A., Politechnica of Bucharest, U., Handley, M., College London, U., . . . catholique de Louvain, U. (2013, 01). *TCP Extensions for Multipath Operation with Multiple Addresses (RFC 6824)*. Retrieved from ietf.org: https://tools.ietf.org/html/rfc6824

F5. (2015, 7 24). *Overview of the TCP Profile*. Retrieved from support.f5.com: https://support.f5.com/kb/en-us/solutions/public/7000/500/sol7559.html

Ford, A., Roke Manor Research, Raiciu, C., Handley, M., University College London, Barre, S., . . . Franklin and Marshall College. (2011, 03). *Architectural Guidelines for Multipath TCP Development: RFC 6182*. Retrieved from ietf.org: http://tools.ietf.org/html/rfc6182

harshanatarajan. (2015, 1 16). *Multi-Path TCP on Palo Alto Networks Firewalls*. Retrieved from live.paloaltonetworks.com: https://live.paloaltonetworks.com/t5/Technologies-Articles/Multi-Path-TCP-on-Palo-Alto-Networks-Firewalls/ta-p/61710

Information Sciences Institute: University of Southern California. (1981, September). *Transmission Control Protocol: RFC 793*. Retrieved from ietf.org: https://tools.ietf.org/html/rfc793

Pearce, C. (2015, 6 2). *BSides Knoxville 2015 - Multipath TCP - Breaking Today's Networks with Tomorrow's Protocols*. Retrieved from http://www.securitytube.net/video/13161

Pearce, C., & Thomas, P. (2014). *Multipath TCP: Breaking today's networks with tomorrow's protocols*. Retrieved from Blackhat.com: https://www.blackhat.com/docs/us-14/materials/us-14-Pearce-Multipath-TCP-Breaking-Todays-Networks-With-Tomorrows-Protocols.pdf

Pearce, K. (2014, 10 20). *MPTCP Roams Free (By Default!) - OS X Yosemite*. Retrieved from Neohapsis Labs: http://labs.neohapsis.com/2014/10/20/mptcp-roams-free-by-default-os-x-yosemite/

Rochet, F., Pereira, O., & Bonaventure, O. (2015, 1 15). *Moving Tor Circuits Towards Multiple-Path: Anonymity and Performance Considerations*. Retrieved from

uclouvain.be: http://www.uclouvain.be/crypto/services/download/publications.pdf.96a2ad0c88787dbb.6d756c7469706174685f616e645f746f725f61636d5f666f726d61742e706466.pdf

The SANS Institute. (2015). *Security 503: Intrustion Detection In-Depth.*

Thomas, P. (2014, 8 6). *Neohapsis / mptcp-abuse*. Retrieved from github.com: https://github.com/Neohapsis/mptcp-abuse

VanWagner, S. (2014, 8 10). *How To Set Default Grub / kernel / boot option on Ubuntu GNU/Linux 14.04*. Retrieved from humans-enabled.com: http://www.humans-enabled.com/2014/08/how-to-set-default-grub-kernel-boot.html

10. Appendix

10.1. TCP background

TCP is a connection oriented and reliable protocol. This means that TCP keeps track of the order that data should be passed to the application layer and retransmits data that has been lost or corrupted. TCP performs the ordering and retransmission through the use of Synchronize (SYN) and Acknowledgement (ACK) sequence numbers. The initial exchange of SYN and ACK sequence numbers is known as a three-way handshake, and establishes the concept of TCP session. For additional details beyond what is discussed in this section, please reference RFC 793. Section paraphrased from: (The SANS Institute, 2015).

10.1.1. TCP three-way handshake

A TCP session exists between an IP address and port pair. A host indicates the desire to setup a TCP session by sending a TCP packet with a SYN flag and SYN sequence number to reference the data that this host sends. If the remote host is listening on the corresponding port, the remote host will reply with a SYN flag and SYN sequence number to reference the data that is sent by the remote host as well as and ACK flag and an ACK sequence number to verify the receipt of data sent by the originating host. The

originating host will then send an ACK flag and ACK sequence number to verify receipt of the remote host data. TCP sequence numbers are used to order segments during the reassembly process, and increment based on the number of payload bytes sent. TCP sequence numbers also increment by one when a SYN or FIN flags is set. ACK numbers increment by one to indicate the next expected sequence number. Section paraphrased from: (Information Sciences Institute: University of Southern California, 1981).

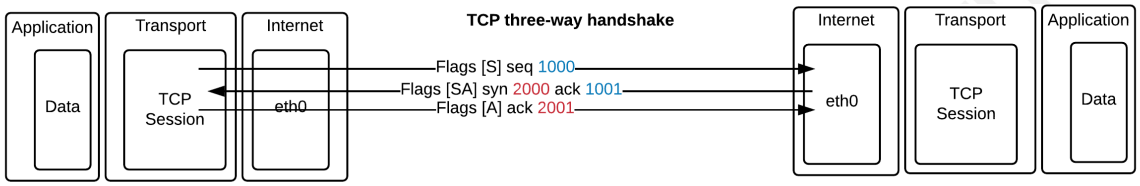

Figure 23 – Putting it all together: TCP three-way handshake

10.1.2. TCP data exchange

Once the TCP three-way handshake has completed and a TCP session has been established, data can be exchanged. The host sending data sets the PUSH flag with the corresponding sequence numbers based on the initial SYN and length of the payload as well as an ACK flag and acknowledge number to verify receipt of the remote host data. The PUSH flag informs the originating host to empty the write buffer and send the data segment(s) to the receiving host. If the data is received, the remote host will respond with ACK flag set and an ACK sequence number to verify the receipt of data sent by the originating host.

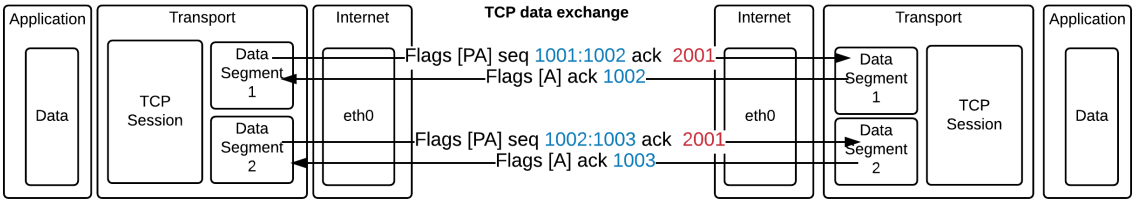

Figure 24 – Putting it all together: TCP data exchange

10.1.3. TCP session termination

After the data exchange is complete, the TCP session can be gracefully or abruptly terminated. In a graceful termination, an originating host will send the FIN and ACK flags set, a SYN sequence number incremented by one to reference data previously sent by the originating host, and a corresponding ACK sequence number. If the packet is

received by the remote host, the remote host will acknowledge the FIN by incrementing the ACK sequence number by one. Since TCP connections are full duplex, the remote host must also set the FIN flag and send the corresponding SYN sequence number that references the previous data sent by the remote host. The originating host will send a FIN flag with an ACK sequence number incremented by one to acknowledge the successful termination of the remote host connection. Once these three steps are completed, a TCP session is successfully terminated. Section paraphrased from: (The SANS Institute, 2015).

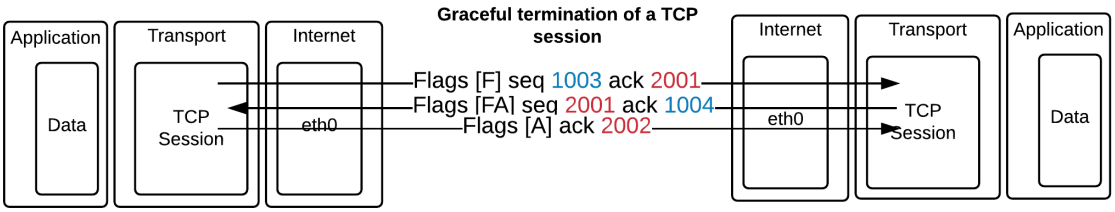

Figure 25 – Putting it all together: Graceful termination of a TCP connection

In addition to a graceful TCP session termination, a session can also be abruptly aborted. In an abrupt TCP session termination, the originating host will send the RST flag with the corresponding SYN and ACK sequence numbers that correspond to the data sent by the originating host and the data sent by the remote host. Once the connection reset is sent, no additional segments will be sent. Section paraphrased from: (The SANS Institute, 2015).

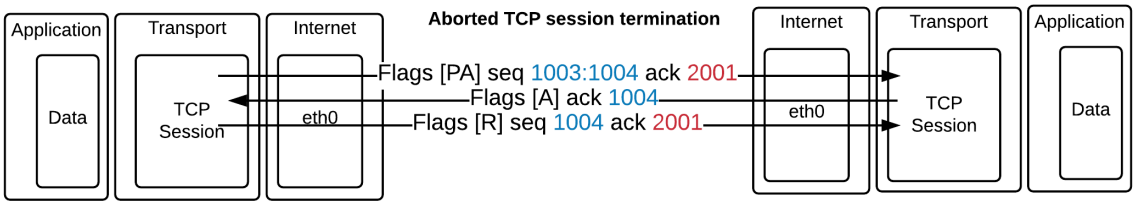

Figure 26 – Putting it all together: Aborted TCP session termination

Beyond TCP connection setup, data transfer, and connection teardown, an understanding of TCP options is import prior to discussing MPTCP. TCP options extend the TCP header by providing a location to store additional parameters. A TCP header without options is 20 bytes long, which allows for 44 bytes of TCP options (Bonaventure, 2013).

Josh Lewis, Joshua.d.Lewis@gmail.com

TCP options are frequently used in TCP connections. Some of the common TCP options include maximum segment size (MSS), window scale, timestamp, selective acknowledgement, and no operation. During the TCP three-way handshake, the originating host sends the TCP options that it supports such as the Maximum Segment Size (MSS), window scale, and selective acknowledgement in the initial SYN. The remote host will acknowledge the TCP options that it supports in the SYN/ACK reply. Based on the response from the remote host, the originating host will specify the options that are supported by both endpoints in the ACK reply to the remote host.

10.2. MPTCP setup

Step 1: Install MPTCP

```
demo@ubuntu:~$ wget -q -O - http://multipath-tcp.org/mptcp.gpg.key | sudo apt-key add –
root@ubuntu:~# echo 'deb http://multipath-tcp.org/repos/apt/debian trusty main' >> /etc/apt/sources.list.d/mptcp.list
root@ubuntu:~# apt-get update
root@ubuntu:~# apt-get install linux-mptcp
```

Figure 27 - Install MPTCP (Barre & Paasch)

Step 2: Boot to the MPTCP kernel

Step number two seems to be missing from the directions provided on multipath-tcp.org. This step modifies grub to boot to the MPTCP kernel by default. The commands in Figure 28 create a copy of the grub file, then identify the submenu string. Copy the submenu string, shown in Figure 29, to a text editor. Identify the MPTCP kernel string, shown in Figure 31, by running the command in Figure 30. Concatenate the submenu string, followed by ">" character, followed by the MPTCP kernel string and copy the concatenated string into the "GRUB_DEFAULT" section of the grub file (reference Figure 32 and Figure 33). Finally, update grub and reboot the host (VanWagner, 2014).

```
root@ubuntu:~# cp /etc/default/grub /etc/default/grub.bak
root@ubuntu:~# grep submenu /boot/grub/grub.cfg
```

Figure 28 - Identify grub submenu

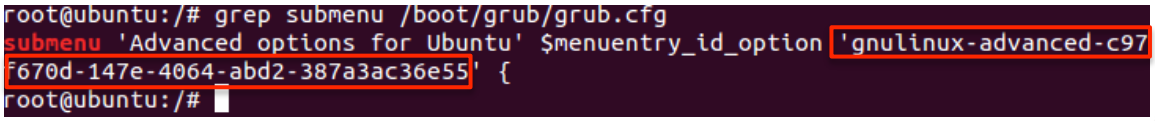

Figure 29 – grub.cfg submenu string

```
root@ubuntu:~# grep gnulinux /boot/grub/grub.cfg
```

Figure 30 – grep for grub.cfg MPTCP kernel string

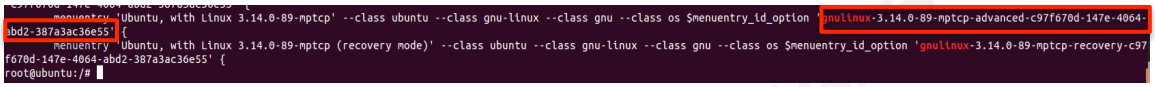

Figure 31 - grub.cfg MPTCP kernel string

```
root@ubuntu:~# vi /etc/default/grub
```

Figure 32 - Edit GRUB_DEFAULT in the grub file

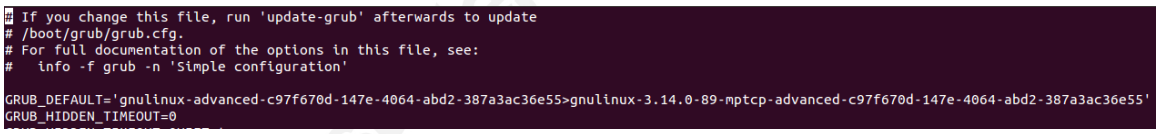

Figure 33 - Copy the concatenated string into the GRUB_DEFAULT line

```
root@ubuntu:~# update-grub
root@ubuntu:~# reboot
```

Figure 34 - Update grub and reboot

Step 3: Configure routing

For a lab environment, routing does not need to be configured for network interfaces on the same subnet. For example, two virtual networks can be utilized, with a virtual machine that has a network adapter in each virtual network. Refer to multipath-tcp.org for additional routing configuration examples and automatic configuration scripts.

```
root@ubuntu:~# echo eth1 ip address 10.10.10.154
root@ubuntu:~# ip rule add from 10.10.10.154 table 1
root@ubuntu:~# echo eth1 ip address 10.10.50.100
root@ubuntu:~# ip rule add from 10.10.50.100 table 2
root@ubuntu:~# echo default gateway for eth1 10.10.10.1
root@ubuntu:~# ip route add 10.10.10.0/24 dev eth1 scope link table 1
root@ubuntu:~# ip route add default via 10.10.10.1 dev eth1 table 1
root@ubuntu:~# echo default gateway for eth0 10.10.50.1
root@ubuntu:~# ip route add 10.10.50.0/24 dev eth0 scope link table 2
root@ubuntu:~# ip route add default via 10.10.50.1 dev eth0 table 2
root@ubuntu:~# echo default gateway for internet traffic
root@ubuntu:~# ip route add default scope global nexthop via 10.10.10.1 dev eth1
```

Figure 35 – MPTCP routing configuration (Barre & Paasch)

10.3. Sample iptables script for MPTCP lab testing

This iptables script is provided for configuration and testing of MPTCP in a lab. Iptables rules should be carefully considered based the appropriate traffic for an enterprise.

```
#!/bin/sh
IPTABLES=/sbin/iptables
IP6TABLES=/sbin/ip6tables
RLIMIT="-m limit --limit 5/min --limit-burst 5"
LOGIPTCPOPTIONS="--log-ip-options --log-tcp-options --log-tcp-sequence"

#Reset existing rules
$IPTABLES -F
$IPTABLES -X

# Clear packet and byte counters
$IPTABLES -Z
$IPTABLES -t nat -Z
$IPTABLES -t mangle -Z

# Set default policies ipv4
$IPTABLES -P INPUT DROP
$IPTABLES -P FORWARD DROP
$IPTABLES -P OUTPUT DROP

# Set default policies ipv6
$IP6TABLES -P INPUT DROP
$IP6TABLES -P OUTPUT DROP
$IP6TABLES -P FORWARD DROP
```

```
#Allow loopback
$IPTABLES -A INPUT -i lo -j ACCEPT
$IPTABLES -A OUTPUT -o lo -j ACCEPT

#Track state
$IPTABLES -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

### Block Rules
#MPTCP log and block
$IPTABLES -A INPUT -p tcp --tcp-option 30 -m state --state NEW $RLIMIT
-j LOG --log-prefix 'IPTBLSDROP: MTCP IN ATTEMPT ' $LOGIPTCPOPTIONS
$IPTABLES -A INPUT -p tcp --tcp-option 30 -m state --state NEW -j DROP
$IPTABLES -A OUTPUT -p tcp --tcp-option 30 -m state --state ESTABLISHED
$RLIMIT -j LOG --log-prefix 'IPTBLSDROP: MTCP OUT ESTAB '
$LOGIPTCPOPTIONS
$IPTABLES -A OUTPUT -p tcp --tcp-option 30 -m state --state NEW -j DROP

### Accept Rules
#DNS
$IPTABLES -A OUTPUT -p udp --dport 53 -m state --state NEW,ESTABLISHED
-j ACCEPT
$IPTABLES -A INPUT -p udp --sport 53 -m state --state ESTABLISHED -j
ACCEPT
$IPTABLES -A OUTPUT -p tcp --dport 53 -m state --state NEW,ESTABLISHED
-j ACCEPT
$IPTABLES -A INPUT -p tcp --sport 53 -m state --state ESTABLISHED -j
ACCEPT

#Incoming SSH
$IPTABLES -A INPUT -p tcp --dport 22 -m state --state NEW,ESTABLISHED -
j ACCEPT
$IPTABLES -A OUTPUT -p tcp --sport 22 -m state --state ESTABLISHED -j
ACCEPT

#Outbound SSH
$IPTABLES -A OUTPUT -p tcp --dport 22 -m state --state NEW,ESTABLISHED
-j ACCEPT
$IPTABLES -A INPUT -p tcp --sport 22 -m state --state ESTABLISHED -j
ACCEPT

#Incoming HTTP
$IPTABLES -A INPUT -p tcp -m multiport --dport 80,443 -m state --state
NEW,ESTABLISHED -j ACCEPT
$IPTABLES -A OUTPUT -p tcp -m multiport --sport 80,443 -m state --state
ESTABLISHED -j ACCEPT

#Outbound HTTP
$IPTABLES -A OUTPUT -p tcp -m multiport --dport 80,443 -m state --state
NEW,ESTABLISHED -j ACCEPT
$IPTABLES -A INPUT -p tcp -m multiport --sport 80,443 -m state --state
ESTABLISHED -j ACCEPT

### Display results
$IPTABLES -nvL
```

10.4. MPTCP proof of concept scanner

Download and extract the zip file from github: https://github.com/Neohapsis/mptcp-abuse.

```
root@ubuntu:~# apt-get install python-pip
root@ubuntu:~# pip install netaddr
root@ubuntu:~# cd /home/demo/Documents/mptcp-abuse-master
root@ubuntu:/home/demo/Documents/mptcp-abuse-master# python
mptcp_scanner.py 10.10.10.239 22
```

Figure 36 - MPTCP poc scanner commands

```
root@demo:/home/demo/Documents/mptcp-abuse-master# python mptcp_scanner.py 10.10.
10.239 22
Testing: 10.10.10.239 ... on local network...  at ARP: 00:0c:29:8e:61:49
 got MPTCP Response from  10.10.10.239 : 22 !...  20
RST Test indicates MPTCP support
****Results:****
        10.10.10.239
                        {22: 'MPTCP (MP_JOIN Verified)'}
root@demo:/home/demo/Documents/mptcp-abuse-master#
```

Figure 37 - MPTCP poc scanner sample output