POLYTECHNIC OF TURIN - STOCKHOLM'S KTH

Faculty of Engineering

Master of Science in Computer Engineering

Master Thesis

MPTCP Security Evaluation

Analysing and fixing critical MPTCP vulnerabilities, contributing to the Linux kernel implementation of the protocol

Advisors:
prof. Antonio Lioy
prof. Peter Sjödin

Candidate:
Fabrizio Demaria

Company tutors
Intel Corporation Inc
Henrik Svensson
Joakim Nordell
Shujuan Chen

March 2016

Acknowledgements

Thanks to...

Summary

Abstract goes here...

Contents

Chapter 1

Introduction

The introductory part explains what MPTCP is and why it is important to study it. Then, the problem statement for this thesis is introduced, to give a good idea of the topics encountered in the paper. Lastly, the methodology followed to solve the problems is presented in the section "Methodology", where the structure of the sections in the paper is also explained.

1.1 Motivation

This section would start with a general introduction of the interconnected world of today, discussing how hardware and software communication has changed in the last decade. The focus of this part is to bring up the multihoming and multipath reality of the infrastructures of today and how this led almost naturally to the MultipathTCP concept. It would be good to cite similar technologies developed before MPTCP (for example SCTP), explaining in which aspects MPTCP is supposed to be a better option. This should include an overview of the real benefits that can be achieved by adopting MPTCP in common appliances (smartphones for example) as well as modern datacenters. It would be good to explain the fact that MPTCP was designed to be as retrocompatible as possible with current infrastructure (lower layer) and applications (higher layer).

1.2 Problem Statement

This part should introduce the content and the main focus of this paper, as well as presenting the objectives of the thesis:

- studying the security implications of adopting MPTCP on current infrastructures;
- listing the known vulnerabilities affecting the current version of the protocol;
- fixing the ADD_ADDR vulnerability of the protocol;
- developing effective and powerful simulation scenarios in order to test MPTCP;
- contributing to the upstreaming of the protocol into the Linux kernel by developing patches and improving official RFC documentation.

1.3 Methodology

This section should contain a short road map with the various step taken to fix the addressed problems and the general methodology adopted for the thesis' work. Perhaps, it is possible to cite here the working environment and the parties involved. This section might end up with an explanation about the structure of the paper.

Chapter 2

Multipath TCP

From now on the discussion becomes more technical. This chapter is all about how MPTCP works from a technical perspective. In this chapter there is no reference about the work carried out during the stage but only information taken from external documentation.

2.1 Transmission Control Protocol (TCP)

This is an explanation on how plain TCP protocol works. Even if there is no need to go into too much details here, this is a necessary starting point from where the MPTCP extension discussion can start (in the next section).

2.2 MPTCP Design

How MPTCP is added on top of TCP (with all the related design aspects) is reported here. This is the first portion of the thesis containing a more in-depth description of how MPTCP works in the networking stack. This specific section might follow closely the introductory portions of the RFC documents regarding MPTCP.

2.2.1 Control Plane

All the MPTCP options used to manage MPTCP sessions are reported and explained here, including all the details on how to set a new session and add/remove subflows.

2.2.2 Data Plane

This part concerns all the MPTCP options used to manage the data flow in a MPTCP session, including how the byte stream is subdivided into different subflow and how the original order of the packets is provided at the receiver.

2.3 MPTCP Deployment

After having explained all the technicalities about the protocol, it is now possible to talk about its deployment status and the problems encountered by pairing MPTCP with current infrastructures. This might seem a bit outside the scope of the thesis, but it is worth mentioning that the *deployment status* and *implementations* are a good indicator of how much MPTCP is important in the Internet community and they are good topics to

motivate the thesis work. Also *middleboxes compatibility* is indeed a fundamental part of the MPTCP security evaluation, being monitoring and detection equipment part of these middleboxes.

2.3.1 Middleboxes Compatibility

This section will be quite technical and it is supposed to list the most important middleboxes and their impact/effect on a MPTCP connection. These boxes include NATs, proxies and firewalls. This part should clearly state why MPTCP widespread adoption is a big challenge.

2.3.2 Implementations

Despite the previously described problematics, MPTCP is a big bet in the IETF community and many implementations have been developed for the most common OSes, listed in this section (with some history notions).

2.3.3 Deployment Status

It should be interesting for the reader to go through some examples of real world's scenarios in which MPTCP is used successfully. Here it is possible to cite some important achievements related to MPTCP (for example the highest throughput ever reached with the new protocol). If available, it would be also good to show some graphs about MPTCP adoption rate or similar.

Chapter 3

MPTCP Security

3.1 Threats Analysis

A complete security evaluation of MPTCP can be subdivided into two main categories:

- A study of the vulnerabilities in the current MPTCP design that can be exploited to carry out flooding or hijacking attacks on an MPTCP session. This is an assessment on how consistently the MPTCP extension would impact the security standards of a plain TCP connection;
- A second perspective is to understand how the new protocol affects the functioning and behaviour of external security gears. This evaluation might include compatibility issues for middleboxes not yet aware of MPTCP [ref section of middleboxes] as well as more fundamental problematics related to security monitoring solutions that wouldn't work anymore with MPTCP: by splitting the logic flow of data into different paths, potentially belonging to different ISPs, it would be much harder to keep track of the content of the transmitted data over the networks. Moreover, the MPTCP ability to reroute traffic on the fly, adding and removing addresses and interfaces, would per se cause major problems with current intrusion detection and intrusion prevention equipment.

This paper focuses on the first point: MPTCP enables data transmission using multiple source-destination address pairs per endpoint and this generates *new* scenarios in which an attacker can exploit the way subflows are generated, maintained and destroyed to perform flooding or hijacking attacks. Flooding attacks are Denial-of-Service procedures that aim at overloading an MPTCP host with connection requests in order to quickly consume its resources. Hijacking attacks aim at taking total control of the MPTCP session, thus being considered the ultimate example of those threats falling in the *active attacks* category (more and this later on, in this section).

MPTCP security mechanism was designed with the primary goal of being at least as good as the one currently available for standard TCP [RFC6181]. The official MPTCP documentation and analysis reports don't cover common threats affecting both TCP and MPTCP, but only the vulnerabilities introduced by the new protocol alone. Nevertheless, it is of paramount importance that the various security mechanisms deployed as part of standard TCP, for example mitigation techniques for reset attacks, are still compatible with Multipath TCP.

Apart from the fundamental objective of keeping MPTCP at least as reliable and secure as TCP, official documents offer another set of requirements mainly related to securing

subflow management in MPTCP [RFC6824bis]. These requirements are:

- Provide a mechanism to confirm that the parties in a subflow handshake are the same as in the original connection setup.
- Provide verification that the peer can receive traffic at a new address before using it as part of a connection.
- Provide replay protection, i.e., ensure that a request to add/remove a subflow is fresh.

MPTCP involves an extensive usage of hash-based handshake algorithms to achieve the required security specifications, as described in chapter 2.

Once the security requirements are clear, it follows a set of related problematics due to the way MPTCP is added to the normal TCP stack. Most notably, the entire behaviour of the protocol relies on the TCP Options field, which is of limited length of 40 bytes. This factor plays an important role in the definition of the security material to be exchanged during an MPTCP session (truncating the HMAC values and tokens is a common technique). Moreover, TCP Options field has been designed to accept any custom protocol extending TCP and for security reasons many middleboxes would discard or modify packets containing unknown options. As a last point, MPTCP approach to subflows' creation implies that a host cannot rely on other established subflows to support the addition of a new one [RFC6182-5.8]; this last requirement follows the *break-before-make* property of MPTCP, that must be able to react to a subflow failure a posteriori by establishing new subflows and automatically sending again the undelivered data. All these considerations define the fundamental boundaries and the context in which the security design of MPTCP has to be developed to meet the requirements.

3.1.1 Threats Classifications

Introducing the support of multiple addresses per endpoint in a single TCP connection does result in additional vulnerabilities compared to single-path TCP. These new vulnerabilities need proper investigation in order to determine which of them can be considered critical and might require modifications in the protocol design in order to meet the required specifications. In order to classify how critical each security threat is, it is a good starting point to define the various typologies of attack according to their requirements, rate of success and what power they can provide to the attacker.

The general requirements for an attack to be executed might be grouped into the following categories:

- *Off-path attacker*: the attacker does not need to be located in any of the paths of the MPTCP connection at any time in order to execute the attack;
- *Partial-time (time-shifted) on-path attacker*: the attacker has to be able to eavesdrop a specific set of information during the lifetime of the MPTCP connection in order to execute the attack. It doesn't need to eavesdrop the entire communication in between the hosts, and the specific direction and/or subflow for the sniffing procedure are attack specific;
- *On-path attacker*: this attacker has to be on at least one of the paths during the entire lifetime of the MPTCP session in order to execute the attack.

We can clearly state that the critical case concerns off-path attacks, which do not require any eavesdrop procedure in order to be executed. In fact, on-path attacks are not considered part of the MPTCP work, since they allows for a significant number of attacks on regular TCP already. A primary goal in the design of MPTCP is not to introduce new ways to perform off-path attacks or time-shifted attacks.

The effects of an attack over an MPTCP connection and the power that the attack can provide to the attacker can be divided into two main categories:

- *Passive attacker*: the attacker is able to capture some or all of the packets of the MPTCP session but it can't manipulate, drop or delay them, and it can't inject new packets in the current session either.
- *Active attacker*: the attacker can pretend to be someone else, introduce new messages, delete existing messages, substitute one message for another, replay old messages, interrupt a communication's channel, or alter stored information in a computer.

The rate of success of a certain attack over a MPTCP connection strongly depends on the specific requirements: two attacks falling in the same categories in terms of attacker eavesdrop capabilities and passive/active typologies might have rather different rates of success. For example, a certain kind of attack might require IP spoofing, thus being unfeasible in a network with ingress filtering [add reference]. There are no general thresholds to define when an attack can be considered a real threat according to the success rate, but this is an important factor to be studied in an attack analysis.

3.2 ADD_ADDR Attack

This paper is mainly focused on studying and testing the ADD_ADDR vulnerability of MPTCP, as well as providing an analysis of the commonly accepted fix and its implementation in Linux kernel. This section describes the attack procedure in details, while other minor residual threats are briefly reported in the next section.

3.2.1 Concept

The ADD_ADDR attack is an *off-path active attack* that exploits a major vulnerability in the MPTCP version 0 design [ref to RFC version 0]. As previously mentioned, the attacks falling into this category are usually the most critical and can easily jeopardize the protocol security requirements. With the current MPTCP model, an attacker can forge and inject an ADD_ADDR message into an MPTCP session to achieve a complete hijacking of the connection, placing itself as a man-in-the-middle. Being this an off-path attack, the attacker can *conceptually* send the forged ADD_ADDR message from anywhere in the network (as allowed by routing), with no need to be physically close to the victim machines. At the end of the attacking procedure, the attacker will be able to operate in any way on the ongoing data transmission, with no clear warning given to the original parties involved in the MPTCP session. If no protection system is used at the application layer (like data encryption), the attacker can eavesdrop all the information and even modify or generate the exchanged content. The attack vector enabled by such exploit is huge.

The culprit is indeed the format of the ADD_ADDR option, whose behaviour will be changed with the new ADD_ADDR2 for the very purpose of fixing this vulnerability [last chapter ref]. This vulnerability is entirely related to the MPTCP design, and due to its

characteristics it is considered a blocking issue in the MPTCP progress towards Standard Track [7430].

3.2.2 Procedure

Let's consider a scenario in which two machines, host A and host B, are communicating over an MPTCP session involving one or more subflows. The attacker is called host C and it is operating remotely with no eavesdrop capabilities. The attacker is using address IPC and targeting a single MPTCP subflow between host A (address IPA and port PA) and host B (address IPB and port B). The scenario is reported in figure 3.1.

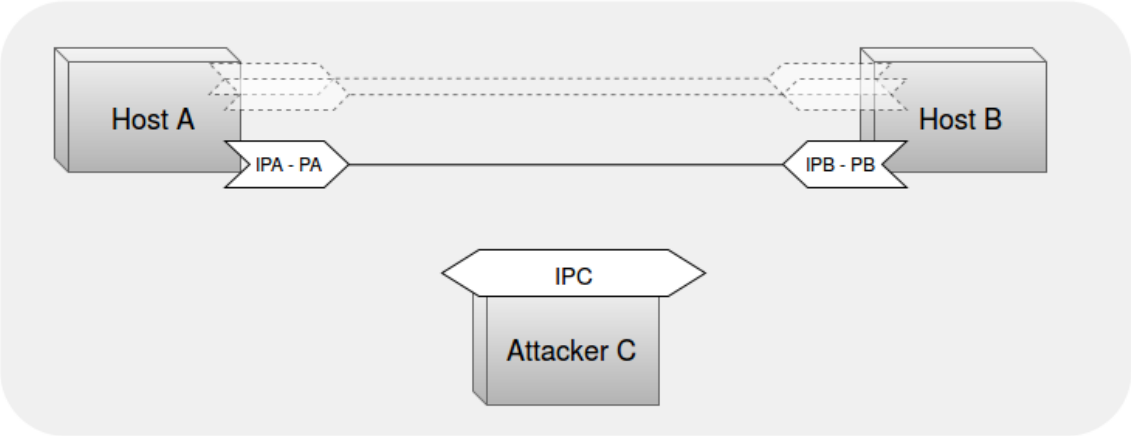

Figure 3.1: Attack scenario

Here is reported the step-by-step procedure to carry out the ADD_ADDR attack:

1. The first step performed by the attacker is to forge an ADD_ADDR message as follows: it is an ACK TCP packet with source address IPA, destination address IPB and the advertised address in the ADD_ADDR option is IPC. The ADD_ADDR option also contains the *Address ID* field, that the attacker can set to a number high enough in order not to collide with existing identifiers for the ongoing subflows between hosts A and B. The format of all the various MPTCP option can be found in chapter 2. The forged packet is then sent to host B.

2. Host B will process the forged packet as a legit request by host A of advertising a new available interface with address IPC. This most likely triggers the creation of a new subflow towards the new IP address, meaning that host B sends a SYN+MP_JOIN packet to the attacker. This packet contains all the security material needed in the first phase of the MP_JOIN three-way handshake, and the attacker does NOT need to operate over that portion of data: the attacker C simply manipulate the SYN+MP_JOIN packet by changing the source IP to IPC and the destination IP to IPA; then, it forwards such packet to host A.

3. Host A will process the incoming packet as a legit request by host B of starting a new subflow from host B's new available interface having address IPC. All the required information is present in the MP_JOIN option, like the token of host A that identifies the specific MPTCP session to which attach the new subflow to. Host A computes all the needed parameters (like a valid HMAC value), generate the SYN/ACK+MP_JOIN packet and send it to IPC. The attacker, similarly to the previous steps, manipulate the IP addresses of the packet from A by changing the

source endpoint from IPA to IPC and the destination endpoint from IPC to IPB. At this point, attacker C sends the packet to host B.

4. All the parameters in the received packet looks correct to host B, which replies with an ACK+MP_JOIN packet to attacker C. The attacker changes the source address to IPC and the destination address to IPA and sends the modified packet to host A. Upon acknowledge reception, host A will verify all the parameters in the packet (which will be correct since properly calculated by host B), and create a new subflow towards IPC. At this point the attacker has managed to place itself as man-in-the-middle.

5. As a further, optional step, the attacker can send RST packets to the other subflow in order to close them thus being able to perform a full hijack of the MPTCP session between host A and host B. The attacker can now operate upon the connection in any possible way, modifying, delaying, dropping, forging packets between the two parties.

By exploit the ADD_ADDR option, the attack procedure is relatively straightforward. Albeit there are some important requirements and limitations that consistently limit the rate of success of such attack, which are discussed in the following section.

3.2.3 Requirements

A first, basic prerequisite needed by the attacker to inject the ADD_ADDR message into an ongoing MPTCP session is to know the IP addresses and port values adopted by host A and host B for the targeted subflow. It is reasonable to assume that the IP addresses are known. In a typical client-server configuration, the server's port for a certain application protocol is fixed and can be assumed to be known, too. For the client counterpart, the port value can cause problems in the presence of protection techniques like port randomization [ref]: in these cases the attacker has to start a guessing procedure whose rate of success also depends on the ephemeral port range employed [ref].

The knowledge about the above-mentioned four-tuple is a basic requirement for obvious reasons, but knowing the endpoint details is not enough to inject valid packets into an ongoing TCP session (that, in this case, can be also seen as an MPTCP subflow session): these packets have to contain SEQ and ACK sequence numbers that are compatible with the current ones within the stream. SEQ and ACK values are used in TCP to provide reliable, in-order transmission of data as well as services related to flow and congestion control [ref]. A very common protection technique is to randomize those 32-bit values at TCP connection setup, forcing the attacker (who acts off-path) to blindly guess them. TCP provides a window mechanism to deal with possible transmission's unalignments: at any given time, the accepted ACK values are those between the last ACK received and the same value plus the receiving window parameter. As a result, the number of packets to be sent in the attempt of guessing the right SEQ and ACK values and consequently the rate of success of the attack are strongly influenced by the TCP receive windows size at the targeted TCP host.

The requirements listed so far all pertain to the underlying TCP protocol. The only MPTCP specific parameter that can cause the failure of the ADD_ADDR attack procedure is the Address ID field in the option. The purpose of this value has been previously explained, and it doesn't actually offer an overall protection improvement. It is enough for the attacker to chose an ID value that is not in use by other subflow in the MPTCP

session. In usual scenarios with a relatively limited number of subflow with the MPTCP session, applying a random value to this field should work just fine.

Moving away from the inner parameters evaluation and taking into consideration external protection mechanisms, it is worth mentioning that the attacker has to be able to manipulate and forge packets, including changing their source address field. This process, known as IP spoofing [ref], is a well known technique for which protection technologies have been developed, most notably the ingress filtering [2827] or source address validation [6056]. However, these methods are not vastly deployed and cannot be considered a sufficient mitigation for the ADD_ADDR vulnerability [ref on inngress filtering usage].

Lastly, the attacker has to be able to direct the malicious ADD_ADDR packet to a host that is actually capable of starting a new subflow, namely the client in a client-server model. The current Linux kernel implementation prohibits the server to instantiate a new subflow and only the client does so.

3.3 Additional Threats

In this section are presented the other residual threats under analysis by the IETF community at the time of writing. They all fall into two main kinds of attacks: flooding attacks and hijacking attacks.

3.3.1 DoS Attack on MP_JOIN

This kind of DoS attack would prevent hosts from creating new subflows. In order to be executed, the attacker has to know a valid token value of an existing MPTCP session. This 32-bit value can be eavesdropped or the attacker has to guess it.

This attack exploits the fact that a host B receiving a SYN+MP_JOIN message will create a state before answering with the SYN/ACK+MP_JOIN packet. This means that some resources will be consumed at the host to keep in memory information regarding this connection request from the other party; in this way, when the host B receives the third ACK+MP_JOIN packet, it can correctly associate it to the initial request and complete the handshake procedure. The creation of such state is required because there is no information in the ACK+MP_JOIN packet that links it to the first SYN+MP_JOIN request, so it is up to the host to remember all the ongoing requests. An attacker can exploit this by sending SYN+MP_JOIN packets to a host without providing the final acknowledge packets. This can be done until the attacked host runs out of available spots for initiating additional subflows. The initial number of such available spots depends on the implementation and configuration at the host machine.

This attack can be exploited to perform a typical TCP flooding attack. This is the perfect example of how MPTCP might introduce new vulnerabilities that might affect the underlying TCP protocol. SYN flooding attacks for TCP have been studied for many years and current implementations use mitigation techniques like SYN cookies [reference] in order to allow stateless connection initiations. But each SYN+MP_JOIN packet received at the host would trigger the creation of an associated state, while this is not the case for the attacker machine that can simply forge these packet in stateless manner. Exploiting this unbalance in resource utilisation is referred to as *amplification attack*.

A possible solution to this problem is to extend the MP_JOIN option format to include the information required to identify a specific request throughout the 3-way handshake, without requiring hosts to create associated states.

3.3.2 Keys eavesdrop

An attacker can obtain the keys exchanged at the beginning of the MPTCP session, exploiting the fact that those are sent in clear. This is in fact a partial-time on-path eavesdropper attack, whose success would enable a vast set of attacking scenarios, even if the attacker itself has moved away from the session after sniffing the aforementioned keys. The keys associated to an MPTCP session are sensitive pieces of information, used to identify a specific connection at the hosts and used as keying material for all the HMAC computations in the protocol. With such pieces of information an attacker can potentially execute a connection hijacking. This problem is encountered again when analysing the ADD_ADDR attack, Section 3.2.

Possible solutions have been proposed to protect the keys, but these are outside the scope of this paper.

3.3.3 SYN/ACK attack

This is a partial-time on-path active attack. An attacker that can intercept and alter the MP_JOIN packets is able to add any address it wants to the session. This is possible because there is no relation between the source addresses and the security material in the MP_JOIN packets. But securing the source address in MP_JOIN is not feasible if MPTCP is supposed to work through NATs: these middle-boxes operate exactly as described in this attack procedure.

Possible solutions have to reside on a different layer, perhaps securing the payload as a technique to limit the impact of such attack in a MPTCP session.

Chapter 4

ADD_ADDR Attack Simulation

Since the first part of the thesis work has been devoted to build the attacking tool to reproduce the exploitation of the ADD_ADDR vulnerability, an entire chapter is dedicated to this topic. This work can be a valuable reference for future simulation setups, involving MPTCP or any other networking protocol.

4.1 Environment Setup

Here it will be explained what UML virtual machines are and why they were good candidates for the simulation tests. The setup procedure is also reported here, with graphs to visually show the simulation's network scenario. The Scapy tool is also presented here. This was a great program to manipulate and forge packets.

4.2 Attack Script

This part is supposed to explain how the Scapy attacking script actually replicates all the steps needed to execute the ADD_ADDR attack. The steps were explained from a theoretical point of view in the previous chapter. Some code snippets of the attacking script can be present in this section. Some details on the problems encountered in building the script and how they have been solved can be valuable material, too.

4.3 Reproducing the Attack

This section is a step-by-step tutorial on how to call the attack script and reproduce the attack. It mainly contains the README file in the GitHub repository where the script can be retrieved. Screenshots and/or Whireshark output can be added to show the actual behaviour of the attack over a netcat connection.

4.4 Conclusions

A final evaluation of the experiment is needed. First, it is important to emphasize the limitations of the adopted tool and how this simulation differs with respect to real scenarios. Nevertheless, it is crucial to explain the value of such experiment, that indeed proves the feasibility of the ADD_ADDR attack and better justify the development of a fix, which is the main topic of the next chapter.

Chapter 5

Fixing ADD_ADDR

This chapter is related to the second and most important part of the work performed during the Master Thesis work at Intel. The ADD_ADDR2 option is developed and added to the current MPTCP implementation for the Linux kernel in order to fix the vulnerability.

5.1 The ADD_ADDR2 format

The actual new format and the reasons why it fixes the vulnerability of ADD_ADDR are reported here. Various discussions can be added to explain why this is believed to be the best way to fix the problem. This part doesn't yet include any implementation details and/or code snippets.

5.2 Implementing ADD_ADDR2

An introductory section that shows the main architectural aspects of how MPTCP has been merged into the TCP code and the TCP modules inside the kernel. Here it starts the part with all the details about the implementation of ADD_ADDR2 in the kernel, as part of the work developed during the stage. Code snippets have to be added here. The following subsections are the side issues and side features that have been elaborated during the thesis work.

5.2.1 Retro-compatibility

Version control mechanism was not present but it is needed to negotiate which format to use in a MPTCP session: ADD_ADDR or ADD_ADDR2.

5.2.2 Port Advertisement

Port advertisement in ADD_ADDR is possible according to RFC specifications but it was not part of the implementation at the beginning of the thesis work, so it has been added.

5.2.3 IPv6 Considerations

Longer addresses brought some issues related to TCP option fields limitations.

5.2.4 Crypto-API in MPTCP

A major problem was how to deal with the new hashing requirements introduced by ADD_ADDR2. Extending the current MPTCP hashing function to deal with input messages of arbitrary size is a first point to explain. The second part has to deal with the whole analysis related to adopting the kernel CRYPTO APIs to calculate the HMAC values in MPTCP and why this is not advisable.

5.3 RFC Contributions

Another minor part of the thesis work on MPTCP is related to some small contributions to the official RFC documentation.

5.4 Experimental Evaluation

This part should include performance analysis regarding the new format introduced with ADD_ADDR2. A discussion on how the new format (and all the other modifications introduced with the patches) could impact any aspect of the protocol should be present in this section.

Chapter 6

Conclusions

6.1 Related Work

References to all the related work, including all the efforts to make MPTCP secure and stable, can be reported here.

6.2 Future Work

Here goes the list of the next steps to be taken care of in terms of MPTCP security, in order to facilitate the protocol's upstream and its widespread deployment. This section can also include a more specific discussion on the aspects to be still analysed regarding the new format ADD_ADDR2.

6.3 Final Thoughts

Some final conclusion.

Appendix A

An appendix

Appendix content goes here...