TLS MITM attack on the Ripple XRP Ledger

Wolfgang Bubberman[1] , **Stefanie Roos**[1] , **Satwik Prabhu Kumble**[1]
[1]Delft University of Technology

Abstract

The Ripple XRP Ledger network hosts a cryptocurrency called XRP which uses the TLS protocol to send messages between nodes. It is crucial that the network is tested against attacks to ensure its security. The Ripple XRP Ledger could be vulnerable to a Man-in-the-Middle (MITM) attack. This MITM attack is an attack which can intercept and modify data by posing as intended receiver of the message. Potentially, this could have major implications as message content could be intercepted and used maliciously. Therefore this paper aims to answer the question: "Is it possible to conduct a TLS MITM attack on the XRP ledger to gain access to message content and how does such an attack affect the performance of the system?". The research done in this paper concluded that it is possible to conduct a TLS MITM attack on the XRP Ledger. The impact on performance is a delay in the travel time of a message. However, how big this delay is, is dependent on the network the attack is executed on. In the tested environment it was roughly adding a 69.2% delay to a message. Future research is necessary to be able to modify the message content and to confirm if the larger the delay added by the MITM, the higher the probability that the MITM is detected.

1 Introduction

Since the initial proposal for Bitcoin in 2008 [11] cryptocurrencies and blockchains have become more widely used over the past decade and a half. The XRP ledger is one of these decentralized networks used for transferring digital assets. It is the home of the cryptocurrency XRP. XRP forms a bridge between the different currencies in use worldwide. The goal of XRP is to contribute to the Internet of Value: a world in which money moves the way information does today [20]. The usage of encrypted channels in decentralized networks is critical to send data from one node to another securely. In the ripple XRP ledger network, the nodes make use of Transport Layer Security (TLS). To get a better insight in how the platform works and to find bugs that went previously undiscovered, a Man-in-the-Middle (MITM) attack can be executed to discover what kind of data and messages are transferred via these encrypted channels. Logically from this we can conclude that if someone is able to perform a MITM attack, a lot of useful information can be gathered regarding the weaknesses of the XRP Ledger, which could be used to exploit or improve the XRP Ledger. Therefore the purpose of this research project is to analyse the impact and possibility of a TLS MITM attack on the Ripple XRP Ledger, which makes the research question the following: "Is it possible to conduct a TLS MITM attack on the XRP ledger to gain access to message content and how does such an attack affect the performance of the system?"

Whilst there is a lot of information and research done and ongoing about MITM attacks by authors such as Nayak and Samaddar [12], and the employment thereof on TLS networks as explained by Prentow and Krarup [17], there has not been any published research regarding a TLS MITM attack on the Ripple XRP Ledger. Therefore this paper, which is about the research of a TLS MITM attack on the Ripple XRP Ledger and the impact thereof, should provide information that fills in this aforementioned gap in the knowledge about TLS MITM attacks and help other research regarding the Ripple XRP Ledger and its security.

In this paper it will be shown that, by a proof of concept, it is possible to conduct such a TLS MITM attack on the Ripple XRP Ledger, and it has a 69,2% delay performance impact on the messages it proxies.

The next section, section two, is about the background information on MITM attacks, TLS and Ripple, which will be followed by an explanation of the methodology employed in the research in section three directly afterwards. After that, the contribution made by this paper will be displayed in section four, and the experimental setup and results of the research will be explained and discussed in section five. This will be followed by responsible research to reflect on ethical aspects of the research in section six and a discussion of the contents of the paper in section seven. The conclusion and future work is portrayed in the final section, section eight.

2 Background information

To understand the environment in which the TLS MITM attack takes place, it is important to look at the XRP Ledger (2.1) and TLS (2.2). For a clear explanation of MITM attacks and the effects of such an attack, the subsection about MITM attacks should be read (2.3).

2.1 The XRP Ledger

The XRP Ledger is a decentralized blockchain that hosts the XRP cryptocurrency. To ensure this XRP ledger works, the network makes use of peer-to-peer servers. These servers could belong to anyone [1]. Unlike many other decentralized blockchains, the XRP Ledger allows users to designate their transactions with any currency they prefer in addition to its native currency, XRP.

Every account that is in the XRP Ledger can send XRP to another account and needs to hold a minimum amount of XRP as a reserve to protect the shared global ledger from growing excessively large as the result of spam or malicious usage. This is done to ensure the XRP Ledger does not grow so fast that it outpaces the tempo of technology improvements, so that a commodity-level machine can always run the current ledger on RAM and the full ledger history on disk. This transfer of XRP is direct, without the need of a gateway or provision of liquidity. This makes XRP a useful bridge currency [22].

The XRP Ledger is very different from other systems that host a cryptocurrency as it does not require a "proof of work" or a "proof of stake" like Bitcoin, Ethereum and many other cryptocurrencies do [7]. The XRP ledger employs a consensus algorithm where every participant has an overlapping set of trusted validators and those trusted validators efficiently agree on which transactions happen in what order. Reaching consensus is done in rounds where each participant of this consensus round proposes a set of transactions they claim to have seen. Shortly after this, participants then come to a consensus by deferring each transaction that does not have a majority support and accepting each transaction that does have a majority support. Slight majorities tend to become major majorities and slightly minorities tend to be quickly universally rejected and deferred by all participants [24].

To prevent consensus from failing and to reduce the overlap needed between validators, the threshold needed to accept transactions increases over time. However, the consensus still fails sometimes, the probability of this happening can be made extremely low but cannot be fully negated. This is solved by the XRP Ledger consensus protocol by having participants post the fingerprint of the ledger that should be the next fully validated ledger. These fingerprints are then exchanged by all participants. There are three possibilities after this happens:

1. They either see that the super-majority found the same ledger and then fully validate and continue with this ledger.
2. They see that they concluded on a different ledger than the super-majority, they realise this and accept the ledger of the super-majority.
3. They cannot conclude on a super-majority for any ledger and have to discard and do another round to reach consensus on any ledger.

The first two cases are completely harmless and to be expected, the network reaches consensus and continuation of operations can be done. The third case leads to a loss of efficiency but is extremely rare and consensus after that failed round should be less likely to fail again [21].

An interesting and key fact about these participants is that they operate using TLS, something which is critical for the contents of this paper and is discussed in the next subsection.

2.2 TLS

The Transport Layer Security (TLS) is a protocol designed for setting up a secure channel between two communicating parties, the client and the server. The protocol that is used for communication with TLS is Hypertext Transfer Protocol (HTTP) [19] most of the time, but other communication protocols can be used as well. TLS is based around the concept of public and private keys with the added security of certificates. Secure Socket Layer (SSL) is the predecessor of TLS and the original encryption protocol developed for the underlying transport layer but is now replaced with TLS. It is important to note that SSL has been replaced by TLS since 1999 [14] but the name is still widely used in the literature regarding TLS.

To initiate a communication session that uses TLS for its encryption, a TLS handshake needs to take place. This handshake takes places when the client queries the server and has already established a Transmission Control Protocol (TCP) connection with the server [18]. During this handshake, the client and server will decide the following [3]:

1. Which version of TLS to use
2. Which cipher suite to use [26]
3. Authenticate the identity of the server via the server's public key and the SSL certificate authority's digital signature
4. Generate session keys to use encryption for communication after the handshake is complete

Due to the nature of different key exchange algorithms and cipher suites used, not every TLS handshake is the same. However, the most widely used key exchange algorithm goes as follows:

1. The client initiates the handshake by sending a **client hello** message, this message includes which version of TLS and the cipher suites the client supports and the client random.
2. The server replies to the client by sending a **server hello** message, which contains the server's SSL certificate, the servers cipher suite and the server random.
3. The client responds by verifying the server's SSL certificate with the certificate authority that issued it to **authenticate** the server.
4. After authenticating the server, the client sends a **premaster secret** to the server which is encrypted with the public key of the server and can only be decrypted with the private key of the server.
5. The server **decrypts** the premaster secret using the private key.
6. With both client and server having the client random, server random and the premaster secret, they both can generate the **session keys**.

7. The client sends a **finished message** to the server that is encrypted with the session key.
8. The server sends a **finished message** to the client that is encrypted with the session key.

After this the handshake is completed and communication can continue using the generated session keys [3].

2.3 Man-in-the-middle attacks

A Man-in-the-Middle attack, or MITM attack for short, is seen as an attack where the attacker secretly acts as a middle man between two parties who believe they are communicating directly with each other. The MITM attacker can modify and/or monitor the communications [25]. Two of the main characteristics of a MITM attack are that they target the associations between the communicating parties and that they represent active attacks as opposed to passive attacks [15] where no modification of data occurs and the target cannot know about its occurrence thereof. This means that a MITM attack always affects the victims in some way or another, be it just a minor slowdown caused by relaying the messages, or by the complete disruption and takeover of the network.

There are several ways a MITM attack can be implemented, some of the most known and widely used ways are Address Resolution Protocol (ARP) cache poisoning and Domain Name System(DNS) spoofing [6]. However, the MITM attacks relevant to this paper are regarding the transport layer. No matter the implementation of a MITM attack, a MITM attack is by definition a very powerful tool, as it either gives the attack the power to do everything the user is allowed to do on the server, or everything the server is allowed to do on the client's side.

The best way to visualise a MITM attack is by seeing the attacker as an entity between the client and the server, where the attacker talks to both client and server separately, and client and server both think they are talking to each other directly. The attacker functionally operates as a proxy. Cryptography becomes of little importance once a MITM attack has successfully been deployed as the MITM attacker knows of all needed data and keys to correctly encrypt and decrypt all messages sent between the client and server. By forwarding all messages and spoofing server and client, a MITM attack can fool almost every authentication system, be it a challenge-response system or a zero-knowledge protocol [2].

Whilst this all might make it look like that MITM attacks are too powerful to be stopped, the TLS protocol is capable of stopping MITM attacks. If the client inspects the servers certificate and finds that the certificate is invalid, the Certifying Authority (CA) of the server is not recognized, or the Common Name (CN) of the certificate does not match with the DNS the client gets a warning and the connection should be dropped [4]. However, the client might ignore the warning and store the certificate, at this point the attacker has won and the client will communicate with the MITM attacker instead of the intended server. Note that this is not the only way to conduct a MITM attack against TLS, the attacker could for example have a valid certificate which does not cause such a warning, or perform SSL stripping [9]. There exist solutions on top of the TLS protocol, but as they are extensions of the protocol, they are not deployed in most instances. Some of these solutions are:

- The Password-based approach [23], which binds the client's authentication over TLS in the server authentication mode.
- The GUI-based approach [16], which uses a specific sequence of images as a password.
- The Token-based approach [8], where both the client generates its access code from a token and the authentication server computes the same code to see if the client is a "real" client.
- The One-Time-Password-based approach, which has many forms but can be explained as the server sending the client a one-time-password to be used for authentication.

3 Methodology

To solve the main research question of this paper, it was key to set up an environment in which a TLS MITM attack could be tested and analyzed. More information on this test-bed environment can be found in the *Experimental setup and Results* section of this paper. This section will talk about the required features of the test-bed environment (3.1), the methodology of testing the feasibility of the TLS MITM attack (3.2) and the methodology of analyzing the impact on performance of the TLS MITM attack (3.3).

3.1 Required features of the test-bed environment

With setting up a test-bed environment for the research to be done, the following features were crucial to achieve for a successful test-bed environment:

The test-bed needed to be able to be used for the development and testing of a TLS MITM attack on a Ripple XRP network. Furthermore, the Ripple XRP network running on the test-bed needed to be manipulable by the administrators and users of the test-bed. The aforementioned test-bed also needed to allow for some form of monitoring that allowed for confirmation of the workings of the attack. The monitoring of the test-bed needed to be sufficient enough to allow for measuring of the performance of the environment, with and without the implemented TLS MITM attack. On top of this, all of these features needed to be stable enough so the TLS MITM attack, when successfully implemented, could be reproduced.

These features were successfully implemented and after implementation allowed for careful and detailed research of the research question as the following two subsections will showcase.

3.2 Feasibility of the TLS MITM attack

After the test-bed was setup in such a way that the above mentioned features were satisfied, the testing and development of a TLS MITM attack could be started. The first part of research questions ponders if it is possible for a TLS MITM attack to be executed on the Ripple XRP Ledger to gain access to the message content. This could be answered by the implementation being either a successful or unsuccessful one.

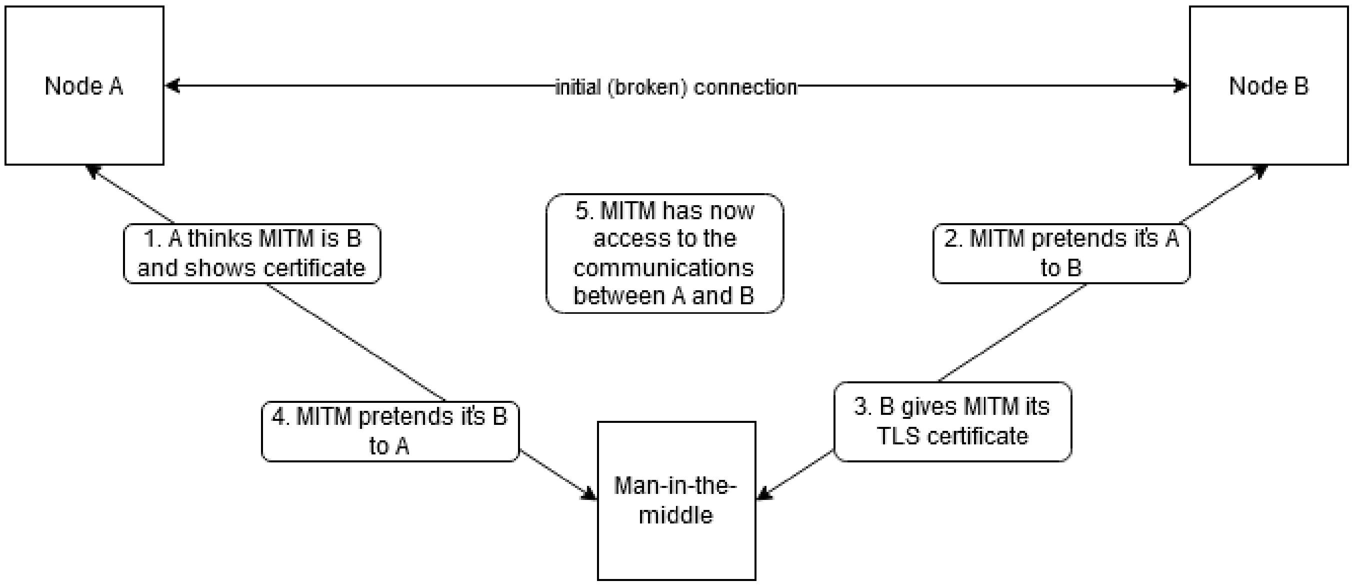
Node A
initial (broken) connection
Node B
1. A thinks MITM is B and shows certificate
5. MITM has now access to the communications between A and B
2. MITM pretends it's A to B
4. MITM pretends it's B to A
3. B gives MITM its TLS certificate
Man-in-the-middle

Figure 1: High-level showcase of the TLS MITM attack in the situation where A wants to send B a message

3.3 Impact on performance of the TLS MITM attack

However, the second part of the research question: ”How does a TLS MITM attack affect the performance of the system?” was not answered with implementing the TLS MITM attack itself. For this examination of the performance of the network was needed, and as the list above describes, the feature that allows this was satisfied with the setup of the test-bed.

By taking samples of the performance with and without the TLS MITM attack on the network, and then comparing the performances after a large enough confidence score was reached with the samples, allowed for an accurate estimation of the impact on the performance of the network by the implemented TLS MITM attack.

This estimation could then be used to give an answer to aforementioned second part of the research question regarding the impact on performance. It should be noted though that this only answers the question regarding the impact of the attack on this implementation and network. The impact on performance estimation should be taken as a rough baseline and suggestion for the impact on performance of a TLS MITM attack, to really answer this question properly, all possible setups of the network needed to be tested on performance impact and to test all possible setups is not a realistic thing to do.

However, the performance of detection of a TLS MITM attack on the XRP Ledger could be researched by analyzing detection methods and testing if a larger variance in performance impact changes in the rate of detection. The main factor that determines the performance is of course the distance between nodes in the network, as in certain network setups nodes might be very close and have a relatively high measurement of performance compared to a network setup with nodes very far from each other.

4 The TLS MITM attack

As this paper is about the feasibility of a TLS MITM attack on the Ripple XRP Ledger and the performance impact of this attack, the execution of a TLS MITM attack to gain access to the TLS message content is crucial. This showcase of a TLS MITM attack on the Ripple XRP Ledger would be an excellent proof by example and a good way to illustrate and research the research question. The decision was made to implement a standalone TLS MITM attack without using a prefabricated alternative. However, there are packages out there, ones like mitmproxy[10] and PolarProxy[13] which can be used in a similar fashion as the suggested own implementation. In this section of the paper, the own implementation will be discussed first (4.1) and the mentioned alternatives will be discussed afterwards (4.2).

4.1 Own implementation

For the showcase it was decided that the private keys and certificates of the nodes that are being attacked had already been retrieved in some way or another. This was to simplify and clarify the attack as much as possible. These keys and certificates could be retrieved via social engineering or some other hack, but this felt out of scope of the research.

There were made some other choices regarding routing of the IP addresses and configuring the network that are more clearly explained in the *Experimental Setup* section of this paper. This includes setting up the cipher suites to be used by the nodes.

For the attack to function, a python3[28] script was run that rerouted the traffic of the two nodes as a proxy, by receiving the messages from one node, decrypting it to read it and then encrypting and forwarding it again to the other node. During this process the MITM acted as node A to node B, and node B to node A. This is more clearly illustrated in the high-level illustration of *Figure 1*.

The decision was made to use python3 for the implementation of the TLS MITM attack as python3 has a very strong and large library for handling TLS connection called *ssl*. This package has very handy functions and structures to setup the connection via the TLS handshake, decide on the cipher suites, select the key and certificates, decryption of messages, and the transport of those messages. It furthermore has strong integration with *asyncio*, another python3 package, which allows for asynchronous sessions to be maintained and handled which is key to the execution of the TLS MITM attack.

With these packages the implementation of the TLS MITM attack became fairly straightforward; when an incoming connection from one of the two targeted nodes got picked up by the MITM, the correct cipher suites, key and certificate were selected to then handle and setup the connection. Afterwards, swift decryption of the message content using the right key could be done, to then later encrypt and forward the message to the other node. This forwarding is done via posing as the node that initially connected to the MITM and setting up a connection with the other node, acting as a client. Then when this connection has been established and the message forwarded, the other node will respond with a response to the initial message. This response can be decrypted, read, re-encrypted and then, via a very similar process, be send back to the initial node using the first connection made from that same node.

Employing this implementation let to the capture of decrypted message content, to be used by the MITM in whatever way possible. An example of a captured, decrypted message is the following:

```
GET / HTTP/1.1
User-Agent: rippled-1.5.0
Upgrade: RTXP/1.2, XRPL/2.0, XRPL/2.1
Connection: Upgrade
Connect-As: Peer
Crawl: private
Network-Time: 643371344
Public-Key: n9M76m17niJoLhxYb2d7g2WzXARP
Su7YBNKT1Pqvx2U3dG4BJeRB
Session-Signature: MEUCIQC22fC9MtSOzOvz0
4uZGdxUrv9bhTkDXEVUDeaZcT0TWAIgJ/fXesYD8
6F9DYuYg4n8to9AnMbxJl942cBnGhWeU3A=
Closed-Ledger: ryaMPaKGk5+l98uVrokdeZpld
5rlPmOrbOPp4z7dLMY=
Previous-Ledger: q4aKbP7sd5wv+EXArwCmQiW
Zhq9AwBl2p/hCtpGJNsc=
```

Note that the hashes are still signed, and thus not manipulable yet. For this to be done, a way needs to be found to modify the signature such that the message can be changed without any party except the MITM knowing, but this is out of the scope of this research.

4.2 Alternatives

There are some alternatives to making a self-made implementation of the TLS MITM attack by using a program like the aforementioned mitmproxy or PolarProxy. Both of these are a good option and have been looked into for this paper. Both programs are very flexible, sophisticated and can be used for the TLS MITM attack on the Ripple XRP Ledger.

Mitmproxy [10]

Mitmproxy is a very extensive program and can be applied in a very similar manner as the suggested implementation discussed above. The website of mitmproxy explains that mitmproxy can be used to intercept, inspect, modify and replay web traffic such as HTTP/1, HTTP/2, WebSockets, or any other SSL/TLS-protected protocols, making it a "swiss-army knife" for debugging, testing and penetration testing. Furthermore, it has a wide range of message types that it supports ranging from HTML to Protobuf and can both encode and decode these messages. It also has a Python API which can be used to extend the workings of the program even further, which was not necessary for this specific research but could be used for later on development such as modification of the message content using a forged signature.

PolarProxy [13]

The official site from NetreSec, the developers of PolarProxy states that: "PolarProxy is a transparent SSL/TLS proxy created for incident responders and malware researchers. PolarProxy is primarily designed to intercept and decrypt TLS encrypted traffic from malware. PolarProxy decrypts and re-encrypts TLS traffic, while also saving the decrypted traffic in a PCAP file that can be loaded into Wireshark or an intrusion detection system (IDS).". As PolarProxy already immediately saves the decrypted traffic to .pcap files it makes it very easy to inspect the traffic and measure the impact on the performance of the network, which is key to this research. It also has four different operation procedures, all of them fairly powerful and useful for malware researchers. However, only the Reverse Proxy and Transparent Forward Proxy can really be applied to this research and of those the Reverse Proxy is the easiest to setup.

The Reverse Proxy mode allows for "... decryption and re-encryption of the proxied traffic while also forwarding important TLS parameters, such as ALPN and SNI." as the website explains. This makes it very much like the suggested implementation of this paper and makes the setup and usage of it fairly straightforward.

5 Experimental Setup and Results

In this section of the paper the experimental setup used to perform the TLS MITM attack and the performance impact it had is discussed (5.1). The results from the research will be presented as well (5.2).

5.1 Experimental Setup

For the experimental setup an Ubuntu server was setup on the TU Delft network where ripple nodes could be hosted and the attack could be performed. For a detailed explanation of how this experimental setup was setup, see appendix A.

The rippled nodes

The nodes were installed in docker containers using the docker rippled image created by the GitHub user WietseWind [29]. The nodes were setup to be one validator node and one normal node, this was done at the time for another ongoing project and this did not interfere with the planned TLS MITM

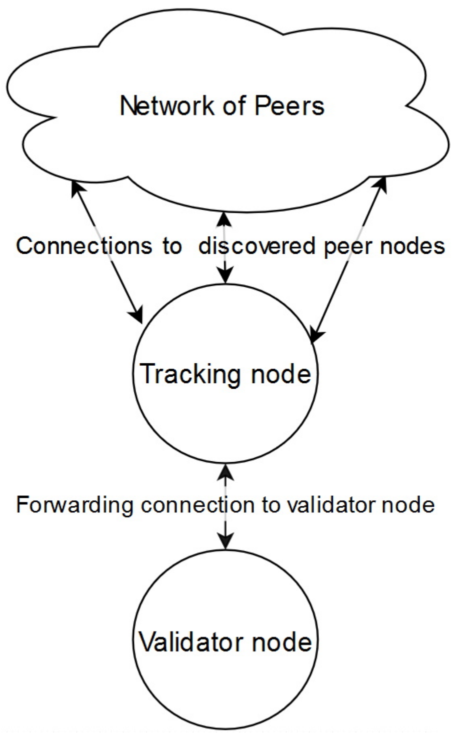
Network of Peers
Connections to discovered peer nodes
Tracking node
Forwarding connection to validator node
Validator node

Figure 2: Network diagram of alternative node network setup

attack. Configuring this was done via the addition of validator keys and a validator token in the configurations file, called RIPPLED.CFG, of the node that would become the validator node. It was also important for the nodes to be in the ripple test network, aptly called *altnet.rippletest*, so this was configured in the configurations file as well. It also needed to be ensured that the nodes had communications with each other, to allow for a MITM attack between the two nodes, this was achieved by configuring the nodes to be peer private. Setting the nodes to be peer private was done by modifying the RIPPLED.CFG again. The cipher suites, keys and certifications of the nodes, which were needed for the TLS communications, were configured by some additions to the RIPPLED.CFG as well. These keys and certifcations were generated using *OpenSSL* to make sure they adhered to the *OpenSSL* standard.

This all led to two nodes that were configured to talk to each other and in a clean test network where no-one would be bothered by the testing and researching to be done.

Another configuration was tested as well, where one of the two nodes had the *peer private* field not set to private to allow for communications with other nodes in the rippled testnetwork. This was done at the time to ensure that the nodes were correctly operating and had not crashed to resolve some bugs. A depiction of this scenario can be seen in *Figure 2*.

The MITM docker container

Once the network was up and running, a space needed to be allocated for the TLS MITM attack itself. This was done by hosting yet another docker container, this time with just a simple Ubuntu image.

To ensure this Ubuntu container could host the attack, a few things needed to be installed. The things that needed to be installed were the following:

- Python3, to be able to run the Python3 script hosting the attack described in the *The TLS MITM attack* section of this paper.
- Vim, the preferred text editor of the developer of this paper, it was needed for editing of the attack script on the fly and other handy file manipulations.
- OpenSSL, the TLS MITM attack hosted in the attack script needed, by definition, to be capable of handling TLS/SSL communications.
- Pip3, the package installer of Python3.

When all of this was installed, the needed Python3 package *asyncio* could be installed using Pip3. Note that the *ssl* package is used as well but is already integrated in the base package of Python3.

Package capturing

Package capturing turned out to be challenging, as the docker network interface behaved unpredictably in the used setup. Thus, running a Wireshark in a docker container to capture all traffic and monitor it on the host itself turned out to be an impractical solution without changing the network structure of the host too much. However, a solution was found by just running *tcpdump*[5] on the host. This allowed for tcpdumps on the docker0 interface on which the nodes and the MITM were running. These tcpdumps were stored in .pcap files, to be then viewed and inspected via Wireshark, either on or off the host.

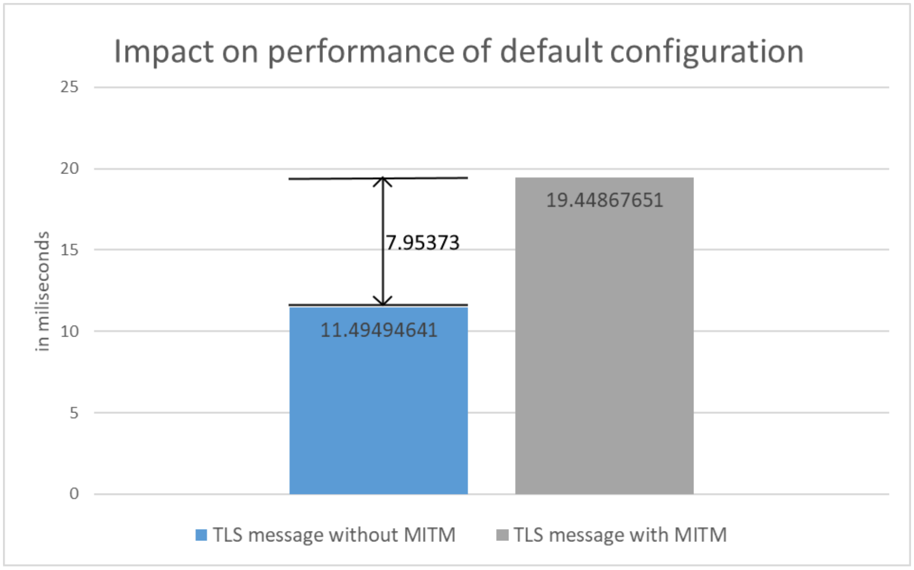

Figure 3: Impact on performance of the default configuration

5.2 Results

After the experimental setup and the development of the attack was completed, configuring the IP address of the MITM in the *ips fixed* section of the RIPPLED.CFG and running the TLS MITM attack on the MITM docker container showed the expected results: message content of the TLS messages between the two nodes. An example of what such a message looks like can be seen in the *The TLS MITM attack* section of this paper.

Thus answering the first part of the research question; yes, it is possible to conduct a TLS MITM attack on the XRP Ledger to gain access to the message content. The research regarding the second part of the research question was a little bit more complicated. What quickly showed was that the location of the Rippled nodes and the MITM mattered in the impact on performance. The closer the MITM was to the Rippled nodes, the smaller the impact on performance. This intuitively makes sense, as the further away the MITM, the longer the detour and thus the longer it takes for a message to be forwarded. Because not every possible configuration of distances and locations can be tested, at the time of writing the only configuration tested is the one described in the experimental setup. However, with the addition of some latency to the docker containers, different configurations can be tested.

The results regarding the impact on the performance of the default configuration as proposed in the experimental setup can be seen in *Figure 3*. The sample size for these numbers of impact was decided upon on a basis of a 0.95 confidence level as is standard in the field of computer science.

These results gave the idea for an interesting hypothesis: “The detection of a TLS MITM attack on the Ripple XRP ledger becomes easier when the impact of performance is larger.” Research from Vallivaara et al [27] into MITM detection methods showed that with current detection methods employed in the industry, it indeed matters how large the added delay of the MITM is. However, at this time, without knowing the detection methods employed by Ripple, it remains unsure whether this hypothesis holds. This remains to be researched further.

6 Responsible Research

This section of the paper is crucial, as it will outline the ethical aspects of the research and tries to clarify what the impacts might be when not adhered to.

As might seem obvious by the nature of MITM attacks, MITM attacks can be used to disrupt systems and networks. To study and understand attackers who employ MITM attacks, research into the prevention and understanding of such attacks exist. The contents of this paper are strictly research related and need to be used only in such a context. The methods in the paper are well documented and are very much reproducible because of this. One can easily setup a test environment using *appendix A* and create the TLS MITM attack itself with the description of the attack in the *The TLS MITM attack* section. Therefore the reader is urged to only use the methods described for research or penetration testing related projects and not for anything nefarious.

Whilst the TLS message content is not manipulable with the current research done in this paper due to the signature that is added by the Rippled nodes, it makes a strong basis for research into this. Therefore the paper might not directly be of any major consequence but further research could be.

7 Discussion

There is an interesting point that has not yet been discussed throughout the paper. When executing the TLS MITM attack on the network as described earlier in the paper, the second node does not respond with normal message content to the initial intercepted message but a 400 BAD REQUEST message more often than not. This message looks like the following:

```
Received as response from 172.17.0.4:
```

```
HTTP/1.1 400 Bad Request
(Failed to verify session)
Server: rippled-1.5.0
Remote-Address: 172.17.0.2
Connection: close
Content-Length: 0
```

This raises two interesting questions; is this the intended response message when a TLS MITM attack is detected on the Ripple XRP Ledger and why does the message sometimes contain valid content like described in the *Results* subsection (5.2) of this paper?

As one might know, or has read in the *Background* section of this paper, there are extensions of the TLS protocol which could detect attacks of all sorts of forms, including MITM attacks. It is unclear at this time what anti-MITM attack method Ripple employs.

An e-mail was sent to Ripple with the findings and the question if this was intended behaviour or perhaps a bug. It was also asked if they have any anti-MITM attack methods they employ. At the time of writing, no answer has been received.

8 Conclusions and Future Work

To summarize and give some conclusions to the research question; "Is it possible to conduct a TLS MITM attack on the XRP ledger to gain access to message content and how does such an attack affect the performance of the system?": Yes, it is possible to conduct a TLS MITM attack on the Ripple XRP Ledger to gain access to message content and there are a few ways of doing so. This can be done with programs like mitmproxy or PolarProxy, but an own solution can be made. The effects such an attack has on the performance of the system is hard to quantify, but it does, intuitively, add some delay to the network and thus brings performance down. The measured performance impact on the tested system was on average 7.95 ms. This added delay is roughly 69.2% of the regular time it takes to send a message.

This does, however, raise some questions regarding the detection of a TLS MITM attack on the Ripple XRP Ledger. Does a larger delay from a TLS MITM attack cause for faster detection? The proposed hypothesis says it does, and ongoing research supports this with the cutting edge MITM detection methods taking delay into consideration. To confirm this hypothesis, future research needs to be done.

Other future research that could build on this paper, might be the research into modifying the session-signature of the decrypted message content intercepted by the TLS MITM attack, this would allow for the complete alteration of message content between nodes which is not yet possible.

References

[1] Edmund L Andrews. Chris larsen: Money without borders. September 2013.

[2] Nadarajah Asokan, Valtteri Niemi, and Kaisa Nyberg. Man-in-the-middle in tunnelled authentication protocols. In *International Workshop on Security Protocols*, pages 28–41. Springer, 2003.

[3] Cloudflare. Tls handshake. https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/, 2020. Accessed: 2020-05-11.

[4] Manik Lal Das and Navkar Samdaria. On the security of ssl/tls-enabled applications. *Applied Computing and informatics*, 10(1-2):68–81, 2014.

[5] The Tcpdump Group. Tcpdump/libpcap. https://www.tcpdump.org/index.html#documentation, 2020. Accessed: 2020-06-01.

[6] Mohammed Abdulridha Hussain, Hai Jin, Zaid Alaa Hussien, Zaid Ameen Abduljabbar, Salah H Abbdal, and Ayad Ibrahim. Dns protection against spoofing and poisoning attacks. In *2016 3rd International Conference on Information Science and Control Engineering (ICISCE)*, pages 1308–1312. IEEE, 2016.

[7] Joshua A Kroll, Ian C Davey, and Edward W Felten. The economics of bitcoin mining, or bitcoin in the presence of adversaries. In *Proceedings of WEIS*, volume 2013, page 11, 2013.

[8] Jing-Chiou Liou and Sujith Bhashyam. A feasible and cost effective two-factor authentication for online transactions. In *The 2nd International Conference on Software Engineering and Data Mining*, pages 47–51. IEEE, 2010.

[9] Moxie Marlinspike. More tricks for defeating ssl in practice. *Black Hat USA*, 2009.

[10] mitmproxy. mitmproxy. https://mitmproxy.org/, 2020. Accessed: 2020-05-31.

[11] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system.(2008), 2008.

[12] Gopi Nath Nayak and Shefalika Ghosh Samaddar. Different flavours of man-in-the-middle attack, consequences and feasible solutions. In *2010 3rd International Conference on Computer Science and Information Technology*, volume 5, pages 491–495. IEEE, 2010.

[13] NetreSec. Polarproxy. https://www.netresec.com/?page=PolarProxy, 2020. Accessed: 2020-05-31.

[14] Rolf Oppliger. *SSL and TLS: Theory and Practice*. Artech House, 2016.

[15] Rolf Oppliger, Ralf Hauser, and David Basin. Ssl/tls session-aware user authentication–or how to effectively thwart the man-in-the-middle. *Computer Communications*, 29(12):2238–2246, 2006.

[16] PassFaces. Authentication. http://www.passfaces.com/enterprise/products/web_access.html, 2009. Accessed: 2020-05-12.

[17] T Siiger Prentow and Mads Vering Krarup. *MITM attacks on SSL/TLS related to renegotiation*. Citeseer, 2011.

[18] Eric Rescorla. *SSL and TLS: designing and building secure systems*, volume 1. Addison-Wesley Reading, 2001.

[19] Eric Rescorla et al. Http over tls, 2000.

[20] Ripple. Our company. https://ripple.com/company, 2019. Accessed: 2020-04-22.

[21] Ripple. Consensus principles and rules. https://xrpl.org/consensus-principles-and-rules.html, 2020. Accessed: 2020-05-10.

[22] Ripple. Xrp. https://xrpl.org/xrp.html, 2020. Accessed: 2020-05-10.

[23] Takamichi Saito, Kiyomi Sekiguchi, and Ryosuke Hatsugai. Authentication binding between tls and http. In *International Conference on Network-Based Information Systems*, pages 252–262. Springer, 2008.

[24] David Schwartz, Noah Youngs, Arthur Britto, et al. The ripple protocol consensus algorithm. *Ripple Labs Inc White Paper*, 5(8), 2014.

[25] Robert Shirey. Internet security glossary, 2000.

[26] Michael Steiner, Peter Buhler, Thomas Eirich, and Michael Waidner. Secure password-based cipher suite for tls. *ACM Transactions on Information and System Security*, 4(2):134–157, 2001.

[27] Visa Vallivaara, Mirko Sailio, and Kimmo Halunen. Detecting man-in-the-middle attacks on non-mobile systems. page 133, 03 2014.

[28] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[29] WietseWind. Docker rippled image. https://github.com/WietseWind/docker-rippled, 2020. Accessed: 2020-05-15.

A Detailed description of experimental setup

This appendix contains a more detailed description of the experimental setup for if one wants to reproduce the research and setup the environment as used in the research.

A.1 The rippled nodes

The nodes were installed in docker containers using the docker rippled image created by the GitHub user WietseWind [29]. For another project that was also ongoing at the time one of the nodes was setup to be a validator node by creating validator keys by running

```
validator-keys create_keys
```

in one of the already running rippled nodes. After this validator tokens were created from these validator keys by running

```
validator-keys create_token --keyfile \
/PATH/TO/YOUR/validator-keys.json
```

and the created token that was of some form like

```
[validator_token]
eyJ2YWZXkiOiI5ZWQ0NWY4NjYyNDFjYzE4YTI3NDd
QzODdjAyMzFmYWE5Mzc0NTdmYT|kYWY2IiwibWFua
c3QiOibXZHdEgyaUNjTUpxQzlnVkZLaWxHZncxL3Z
hYWExwJ3RHdEYklENk9NU1l1TTBGREFscEFnTms4U
bjdNTzQUtxWFlvdUorbDJWMFcrc0FPa1ZCK1pSUzZ
hsSkFm9KYy9hQVpva1MxdnltR21WcmxIUEtXWDNZe
NmluOEhBU1FLUHVnQkQ2N2tNYVJGdQWXk1QXFEZWR
VUSmEydzBpMjFlcTNNWXl3TFZKWm5G3QROD0ifQ==
```

was inserted in the configurations file called RIPPLED.CFG.

To ensure that the nodes did not mess with the main ripple network, the nodes were configured to have

```
[ips]
s.altnet.rippletest.net 51235
```

in their RIPPLED.CFG file and the following in their VALIDATORS.TXT file:

```
[validator_list_sites]
https://vl.altnet.rippletest.net

[validator_list_keys]
ED2648071028052200DA0F312E71FC2C69E \
1552C9C5790F6C25E3729DEB573D5860
```

This replaced some fields that were already configured to connect the nodes to the main network but this was intended to ensure private operations and good testing environment.

It also needed to be ensured that the nodes had communications with each other, to allow for a MITM attack between the two nodes, this was achieved by configuring the nodes to be peer private. Setting the nodes to be peer private was done by modifying the RIPPLED.CFG again, now adding the following fields:

```
[peer_private]
1

[ips_fixed]
ip.of.other.node 51235

[server]
port_rpc_admin_local
port_ws_public
port_peer
```

The cipher suites, keys and certifications of the nodes were configured by some additions to the RIPPLED.CFG *server* field as well. They were the following:

```
[server]
...
ssl_key = /PATH/TO/key.pem
ssl_cert = /PATH/TO/cert.pem
ssl_ciphers = RSA
```

As one might already have thought, for this keys and certifications needed to be generated which was done according to the *OpenSSL* standard:

```
openssl req -x509 -newkey rsa:4096 -keyout \
key.pem -out cert.pem -days 365
```

This all led to two nodes that were configured to talk to each other and in a clean test network where no-one would be bothered by the testing and researching to be done.

Another configuration was tested as well, where one of the two nodes had the *peer private* field not set to private to allow for communications with other nodes in the rippled test-network. This was done at the time to ensure that the nodes were correctly operating and had not crashed to resolve some bugs. A depiction of this scenario can be seen in *Figure 2*.

A.2 The MITM docker container

Once the network was up and running, a space needed to be allocated for the TLS MITM attack itself. This was done by hosting yet another docker container, this time with just a simple Ubuntu image.

To ensure this Ubuntu container could host the attack a few things needed to be installed. This was done with the handy *apt-get*. Installing something would then go as following with *apt-get*:

```
sudo apt-get update
sudo apt-get install NAME-OF-PACKAGE-HERE
```

The things that needed to be installed were the following:

- Python3, to be able to run the Python3 script hosting the attack described in the *The TLS MITM attack* section of this paper.
- Vim, the preferred text editor of the developer of this paper, it was needed for editing of the attack script on the fly and other handy file manipulations.
- OpenSSL, the TLS MITM attack hosted in the attack script needed, by definition, to be capable of handling TLS/SSL communications.
- Pip3, the package installer of Python3.

When all of this was installed, the needed Python3 package *asyncio* could be installed using Pip3. This was done by executing the following on the command-line:

```
pip3 install asyncio
```

Note that the *ssl* package is used as well, but is already integrated in the base package of Python3.

A.3 Package capturing

Package capturing turned out to be a little tricky, as the docker network interface refused to play nice. Thus running a Wireshark in a docker container to capture all traffic and monitor it on the host itself turned out to be an impractical solution without changing the network structure of the host too much. However, a solution was found by just running *tcpdump*[5] on the host.

The tcpdumps were run using:

```
tcpdump -i docker0 -w PATH/TO/FILE.pcap
```

This allowed for tcpdumps on the docker0 interface on which the nodes and the MITM were running. These tcpdumps were stored in .pcap files, as specified by the example above to be then viewed and inspected via Wireshark, either on or off the host.