OST
Ostschweizer
Fachhochschule

RDP Man-in-the-Middle

Bachelorarbeit

Studiengang Informatik
OST – Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Frühjahrssemester 2021

Autoren:	Kevin Moro, Danusan Premananthan, Aynkaran Sundralingam
Betreuer:	Cyrill Brunschwiler
Experte:	Christoph Frei
Gegenleser:	Frank Koch

Abstract

Man-in-the-Middle ist eine Cyberangriffstechnik, bei der sich der Angreifer in den Datenverkehr zwischen zwei Kommunikationsteilnehmern einschleust und beiden vortäuscht, dass sie direkt mit dem jeweils anderen kommunizieren. Das Ziel dieses Angriffes ist es, die Kommunikation zwischen den Kommunikationsteilnehmern abzufangen, mitzulesen oder unbemerkt zu manipulieren.
In dieser Arbeit soll nun die Machbarkeit eines solchen Angriffes auf das Remote Desktop Protocol (RDP), ein Kommunikationsprotokoll von Microsoft, das den Fernzugriff auf Windows-Rechner ermöglicht, untersucht werden. Dabei soll aber die erweiterte Sicherheitsstufe «Enhanced Security NLA» von RDP verwendet werden.
Sollte die Machbarkeitsanalyse zeigen, dass ein Man-in-the-Middle-Angriff auf eine mittels NLA (CredSSP) authentifizierte RDP-Verbindung möglich sein sollte, soll ein Proof of Concept für das von der Firma GoSecure, einem auf Cybersecurity spezialisierten IT-Unternehmen, entwickelte Tool «PyRDP» implementiert werden.

Um die Durchführbarkeit dieses Angriffes beurteilen zu können, musste zunächst das RDP-Protokoll selbst näher untersucht werden. Denn es ist wichtig zu wissen, welche Sicherheitsaspekte RDP unterstützt bzw. verwendet und welche Schritte zum Aufbau einer Verbindung notwendig sind.
Anschliessend wurden relevante Protokolle wie CredSSP, SPNEGO oder NTLM, die bei der vorherigen Untersuchung gefunden wurden, ebenfalls näher analysiert.
Die Machbarkeitsanalyse wurde auf Basis der Theorie und durch die Durchführung von Paketanalysen mit Tools wie Wireshark vorgenommen. Basierend auf diesen Analysen konnte schliesslich die Machbarkeit eines Man-in-the-Middle Angriffes evaluiert werden.

Anhand der durchgeführten Machbarkeitsanalyse konnte festgestellt werden, dass ein direkter Man-in-the-Middle Angriff auf eine durch NLA authentifizierte RDP-Verbindung nicht möglich ist. Denn durch die theoretische Analyse konnte herausgefunden werden, dass das Passwort des RDP-Clients in die Berechnung eines Verschlüsselungsschlüssel einfliesst, der im Authentifizierungsverfahren verwendet wird.
Ein Man-in-the-Middle-Angriff wäre jedoch möglich, wenn der Angreifer bereits im Besitz des Client-Passwortes ist, bevor der Client und der Server das von NLA verwendete Authentifizierungsverfahren durchlaufen.
Um diese Variante zu realisieren, wurde ein alternativer Man-in-the-Middle-Ansatz gewählt. In einem ersten Schritt wird das Kennwort des Clients über einen vorgetäuschten Anmeldebildschirm eingelesen. Mit Kenntnis des Kennwortes kann der Angreifer nun in einem zweiten Schritt eine gültige RDP-Verbindung zum Server aufbauen. Diese wird dann vom Angreifer an den Client weitergeleitet, so dass der Client den Eindruck hat, er sei direkt mit dem Server verbunden.

Inhaltsverzeichnis

1 Allgemeines

1.1 **Änderungsgeschichte**

Datum	Version	Änderung	Autor
23.04.2021	1.0	Initialer Entwurf	Kevin Moro Danusan Premananthan Aynkaran Sundralingam
18.06.2021	5.0	Überarbeitet	Kevin Moro Danusan Premananthan Aynkaran Sundralingam

1.2 **Konventionen**

► Hinweise / Fazit

Commands, Code Snippets, Scripts und Error messages / Exceptions

Querverweise innerhalb des Dokuments sind kursiv dargestellt

Links auf externe Ressourcen sind blau unterstrichen dargestellt

Teil I

Einleitung

2 Management Summary

RDP Man-in-the-Middle

Diplomanden	Kevin Moro, Danusan Premananthan. Aynkaran Sundralingam
Examinator	Cyrill Brunschwiler
Themengebiet	Sicherheit, Software

2.1 **Ausgangslage**

Im Oktober 2020 wurde behauptet, dass das Tool «PyRDP» RDP Verbindungen die mit NLA/CredSSP geschützt sind, intercepten kann. Jedoch wird dafür Schlüsselmaterial vom Server benötigt, was eine starke und etwas kuriose Einschränkung ist.
Die Aufgabe dieser Bachelorarbeit ist es nun, eine mit CredSSP geschützte RDP Verbindung auf die Möglichkeit eines Man-in-the-Middle Angriffes, bei der vorher nicht irgendwelches Schüsselmaterial vom Server gestohlen wird, zu untersuchen. Das Tool PyRDP wird während der Arbeit ebenfalls analysiert und je nach Ergebnis der Analyse noch erweitert.

2.2 **Vorgehen und Technologien**

Um festzustellen, ob ein Man-in-the-Middle Angriff auf eine mit CredSSP gesicherte RDP Verbindung möglich ist, werden als erstes alle involvierten Protokolle für den Verbindungsaufbau und die Authentifizierung genau analysiert. Dafür werden die Dokumentationen der Protokolle und auch Wireshark benutzt. Nach dieser Analyse wird je nach Ergebnis der Analyse ein Proof-of-Concept in «PyRDP» implementiert.
Damit ein allfälliger Proof-of-Concept implementiert werden kann, wird auch der Code von «PyRDP» analysiert und dokumentiert.

2.3 **Ergebnisse**

Die Ergebnisse dieser Arbeit sind die Analyse der involvierten Protokolle und Authentifizierungsmechanismen, sowie die Analyse und Entscheidung über die Machbarkeit dieser Man-in-the-Middle Variante.
Ebenfalls wurde eine leicht abgeänderte Variante des Man-in-the-Middle Angriffes ausgearbeitet und in PyRDP implementiert.

3 Aufgabenstellung

RDP Man-in-the-Middle

Aufgabenstellung BA FS 2021

Datum: Februar 25., 2021
Author: Cyrill Brunschwiler, Compass Security Schweiz AG
Classification: INTERNAL

Compass Security Schweiz AG, Werkstrasse 20, Postfach 2038, CH-8645 Jona
T +41 55 214 41 60, team.csch@compass-security.com, www.compass-security.com

Table of Contents

1 Einführung

Pyrdp ist ein Python 3 Remote Desktop Protocol (RDP) Man-in-the-Middle (MitM) Tool und Bibliothek.

Es enthält einige Werkzeuge:
- RDP Man-in-the-Middle
- Protokollierung der Anmeldeinformationen
- Konsolenbefehle oder PowerShell-Payloads automatisch ausführen
- Übernahme der Steuerung
- Aufzeichnung von Verbindungen
- ...

Im Oktober wurde nun behauptet, dass Pyrdp auch RDP-Verbindungen, die mit NLA/CredSSP geschützt sind, intercepten kann. Bei genauerer Analyse stellten wir aber fest, dass dies mit Pyrdp nur möglich ist, wenn der Angreifer bereits Schlüsselmaterial vom Serversystem gestohlen hat. Ein etwas kurioses Szenario.

2 Aufgabe

Ein MitM Angriff könnte aber trotzdem möglich sein. Wir wollen es deshalb genau wissen und das Protokoll etwas genauer analysieren und falls möglich ein PoC implementieren.

3 Vorgehen

Im Rahmen der allgemeinen Richtlinien zur Durchführung von Studien- und Bachelorarbeiten gemäss eigenem Projektmanagementplan. Dieser Projektmanagementplan ist als Erstes zu erstellen und enthält insbesondere:

- Die Beschreibung des dem Projektcharakter angepassten Vorgehensmodells.
- Eine erste Aufteilung der Aufgabe in gemeinsam und einzeln zu bearbeitende Teile unter Berücksichtigung der vorgegebenen Teilaspekte. Die genaue Aufteilung muss spätestens nach der Technologiestudie (Elaboration) erfolgen.
- Den Projektplan (Zeitplan) und die Meilensteine.

4 Anforderungen

- Analyse der verschiedenen RDP Varianten (Historie)
- Aufzeigen von bereits "gebrochenen" Protokollvarianten
- Studium der Microsoft Security Support Provider Interface (SSPI)
- Studium der SPNEGO, NLA und CredSSP Verfahren
- Analyse von RDP mit NLA/CredSSP
- Analyse der Pyrdp implementation
- Aufzeigen der Machbarkeit oder eben Sicherheit
- Implementation eines MitM PoC basierend auf Pyrdp
- Software Engineering, Requirementsanalyse für Pyrdp (Python)

5 Infrastruktur

Die Arbeiten werden auf den Rechnern der Studenten durchgeführt. Zusätzlich benötigte Software oder Hardware wird bei Bedarf und nach Rücksprache mit Compass Security zur Verfügung gestellt.

6 Erwartete Resultate

6.1 In elektronischer Form:

- Technologiestudien, Know-how Aufbau, Setup Guide
- Software (falls relevant)
    - kompletter Source Code
    - komplette Dokumentation (Use Cases, Klassenmodell, Sequenzdiagramme usw. in UML)
- alle Dokumente
- alle Protokolle

Gemäss der Anleitung der HSR: https://skripte.hsr.ch/Informatik/Fachbereich/Bachelorarbeit_Informatik/

Es muss aus den abgegebenen Dokumenten klar hervorgehen, wer für welchen Teil der Arbeit und der Dokumentation verantwortlich war (detaillierte Zeiterfassung).

7 Termine

7.1 Start/Ende

Termine gemäss Teams

Datum	Task
25.01. bis 18.02.21	Einrichten der Arbeitsplätze in den Labors und Vorbesprechung mit Betreuer/Betreuerin
22.02.2021	Beginn der Bachelorarbeit, Ausgabe der Aufgabenstellung durch den Betreuer/die Betreuerin. 9.6. Die Studierenden geben den Abstract für die Bachelorarbeits-broschüre zur Kontrolle an ihren Betreuer/ihre Betreuerin frei. Die Studierenden erhalten vorgängig vom Studiengangsekretariat die Aufforderung mit den Zugangsdaten zur Online-Erfassung des Abstracts.Die Studierenden senden per Email das A0-Poster zur Prüfung an ihren Betreuer/ihre Betreuerin.Vorlagen sowie eine ausführliche Anleitung betreffend Dokumentation stehen auf Teams zur Verfügung.
15.06.2021	Der Betreuer/die Betreuerin gibt das Dokument mit dem korrekten und vollständigen Abstract der Broschüre zur Weiterverarbeitung an das Studiengangsekretariat frei.Für die Ausstellung der Bachelorarbeiten das A0 Posters per Email bis 10.00 Uhr an das Studiengangsekretariat senden.
18.06.2021	Hochladen aller verlangten Dokumente auf archiv-i.hsr.chAbgabe des Berichts an den Betreuer/die Betreuerin bis 12.00 Uhr
18.06.2021	Präsentation und Ausstellung der Bachelorarbeiten, 16 bis 20 Uhr
16.08.2021 - 10.09.2021	Mündliche BA-Prüfung
01.10.2021	Bachelorfeier

7.2 Zeitplan und Meilensteine

Zeitplan und Meilensteine für das Projekt sind von den Studenten selber zu erarbeiten und zusammen mit dem Projektmanagementplan abzuliefern. Die Meilensteine sind bindend. Der erste Meilenstein ist vorgegeben. Mit den Betreuern werden regelmässige Sitzungen zur Fortschrittskontrolle durchgeführt.

8 Betreuung

Die Arbeiten werden durch Cyrill Brunschwiler betreut.

8.1 Kontakt

Cyrill Brunschwiler, Managing Director, Compass Security Schweiz AG
Weststrasse 50, 8003 Zürich, Switzerland
Werkstrasse 20, 8645 Jona, Switzerland

+41 44 455 6412
cyrill.brunschwiler@compass-security.com
cyrill.brunschwiler@hsr.ch
https://fb.compass-security.com/inbox/hUGXMr2EeZ2V7b

9 Referenzen

- GoSecure announcing MitM of NLA, CredSSP protected RDP connections https://www.gosecure.net/blog/2020/10/20/announcing-pyrdp-1/
- GoSecure pyrdp MitM tool https://github.com/GoSecure/pyrdp/pull/229/files
- Microsoft CredSSP https://docs.microsoft.com/en-us/windows/win32/secauthn/credential-security-support-provider
- Microsoft SPNEGO https://docs.microsoft.com/en-us/windows/win32/secauthn/microsoft-negotiate

10 Unterschriften

Jona, 18. März 2021

Cyrill Brunschwiler | Danusan Premananthan | Aynkaran Sundralingam | Kevin Moro

Teil II

Technischer Bericht

4 Theorie

In diesem Kapitel werden die Begriffe, die für diese Arbeit relevant sind oder im Rahmen der Arbeit auftauchen, kurz erläutert. Dies soll helfen, die vorzunehmende Analyse zu verstehen. Die genauere Analyse dieser wird im nächsten Kapitel beschrieben.

4.1 **Remote Desktop Protocol**

Das Remote Desktop Protocol ist ein Kommunikationsprotokoll von Microsoft, das einen Fernzugriff auf Windows-Rechner ermöglicht. Die Grundfunktionalität von RDP besteht darin, grafische Bildschirminhalte von einem entfernten Server zum Client zu übertragen (Abbildung 1) und die grundlegenden Peripheriefunktionen eines Clients, wie Tastatur, Maus, Druckerkopplung usw., bereitzustellen. Um eine Verbindung auf einen Remote-Server herzustellen, verwendet der Client eine RDP-Client-Software, während auf dem Remote-Server eine RDP-Server-Software installiert sein muss. Die RDP-Client-Software gibt es für Windows, Linux, macOS und für viele weitere Betriebssysteme. Die RDP-Server-Software ist standardmässig in Windows-Betriebssystemen integriert. Die RDP-Server-Software gibt es auch für Unix- und OS X-Systeme. Die Kommunikation in RDP basiert auf mehreren Kanälen und wird standardmässig mit der RC4-Cipher von RSA verschlüsselt. (Reiner, Shaked, 2020)

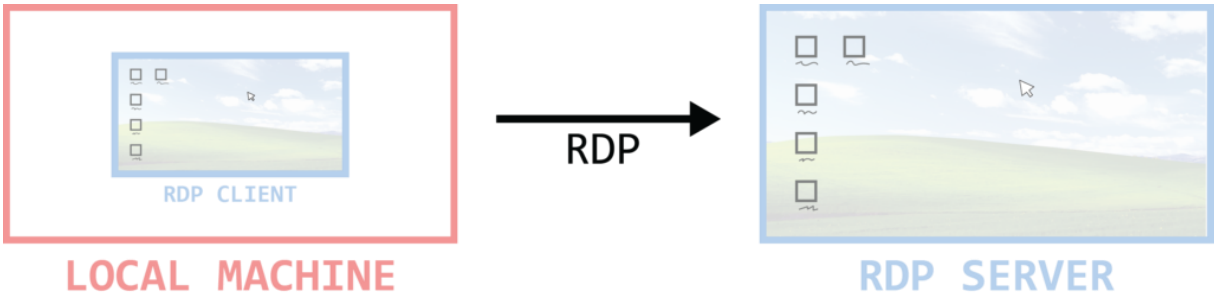


Abbildung 1: RDP Grundfunktionalität (Quelle: (Reiner, Shaked, 2020))

4.2 **Man-in-the-Middle**

Der Man-in-the-Middle-Angriff, kurz «MitM», ist eine Cyberangriffstechnik, bei der sich ein Angreifer in den Datenverkehr zwischen zwei Kommunikationsteilnehmern einschleust und beiden Teilnehmern vortäuscht (Abbildung 2), dass sie direkt mit dem jeweils anderen kommunizieren. Dazu schaltet der Angreifer ein von ihm kontrolliertes System physisch oder logisch zwischen die beiden Kommunikationsteilnehmer. Das Ziel dieses Angriffes ist es, die Kommunikation zwischen den Kommunikationsteilnehmern abzufangen, mitzulesen oder unbemerkt zu manipulieren.

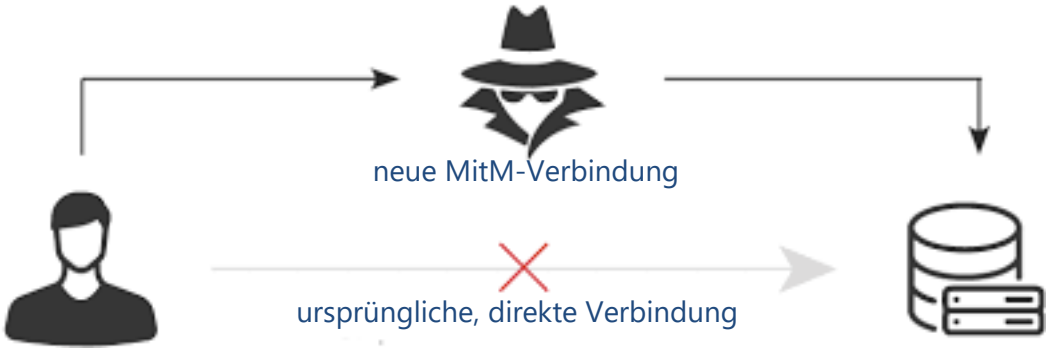


Abbildung 2: Man-in-the-Middle Angriff (Quelle: (Suau, 2019) überarbeitet)

Um einen MitM-Angriff durchzuführen und so den Datenverkehr zwischen zwei oder mehreren Systemen zu beeinflussen, verwenden Angreifer verschiedene Techniken, die bei bekannten Sicherheitslücken in der Internet-Kommunikation ansetzen. (IONOS, 2019)

4.3 **PyRDP**

Der Inhalt und die Ausführungen des folgenden Abschnittes beruhen auf folgenden Quellen: (pyrdp, 2021)

PyRDP ist ein Man-in-the-Middle Tool und eine Library für das RDP Protokoll, welches von dem Unternehmen GoSecure, welches sich auf Cybersecurity spezialisiert hat, entwickelt wurde. PyRDP ist in Python geschrieben. Dieses Tool wurde im Jahr 2018 für Malware- und Pentest-Forschungszwecke entwickelt.

4.3.1 Features

PyRDP verfügt über folgende Tools:

- RDP Man-in-the-Middle:
    - Protokollierung der Credentials beim Verbindungsaufbau
    - Stehlen der kopierten Daten aus der Zwischenablage
    - Von übertragenen Dateien über das Netzwerk wird eine Kopie gespeichert
    - Freigegebene Laufwerke werden im Hintergrund durchsucht und lokal gespeichert
    - Speichert Wiederholungen von RDP-Verbindungen und kann später angesehen werden
    - PowerShell-Payloads oder Konsolenbefehle werden bei neuen Verbindungen automatisch ausgeführt
- RDP-Player:
    - Live RDP-Verbindungen, welche vom MitM kommen, sehen
    - Wiederholungen von RDP-Verbindungen anzeigen
    - Kontrolle über aktive RDP-Sitzungen übernehmen und dabei ihre Aktionen verbergen
    - Auflisten der zugeordneten Laufwerke des Clients und Herunterladen von Daten während aktiver Sitzungen
- RDP-Zertifikat-Cloner:
    - Erstellen eines selbstsignierten X509-Zertifikates mit denselben Feldern wie das Zertifikat eines RDP-Servers

4.4 **Network Level Authentication**

Network Level Authentication, kurz NLA, ist eine optionale Windows-Funktion, die vom RDP Protokoll verwendet wird. Diese Funktion kann auf dem RDP Server aktiviert werden. Die Aktivierung dieser Funktion erfordert, dass der RDP-Client sich vor dem Verbindungsaufbau mit dem RDP-Server authentifizieren muss. Dabei werden die Anmeldedaten des RDP-Clients an einen clientseitigen Security Support Provider (4.6 Security Support Provider) weitergeleitet, bei dem der Client authentifiziert wird. Die Authentifizierung mittels NLA basiert auf der Netzwerkebene. NLA wurde mit der RDP-Version 6.0 eingeführt. (Wikipedia, 2021)

4.5 **Security Support Provider Interface**

Security Support Provider Interface (SSPI) ist eine Komponente der Windows-API. SSPI ist das Grundgerüst für die Implementierung von sicherheitsrelevanten Operationen, wie z.B. die Durchführung von Authentifizierungen. Dabei unterliegen mehrere Security Support Provider (4.6 Security Support Provider) dieser Schnittstelle. Die SSPI wird durch NLA aufgerufen und wählt aufgrund der Bedingungen des Anwendungsprotokolls, in dieser Arbeit z.B. RDP, den zu verwendenden Security Support Provider. (Wikipedia, 2021; Microsoft, 2018)

4.6 **Security Support Provider**

Security Support Provider ist eine dynamische Programmbibliothek, die eine oder mehrere Sicherheitspakete für Anwendungen zur Verfügung stellt. Unter Windows existieren mehrere SSPs für die Durchführung von Sicherheitsoperationen. Das RDP-Protokoll nutzt für die Authentifizierung über NLA das CredSSP-Paket. Dieses wird daher bei der Durchführung von RDP-Authentifizierungen automatisch vom SSPI ausgewählt. (Wikipedia, 2021; Microsoft, 2018)

4.7 **TLS**

Der Inhalt und die Ausführungen des folgenden Abschnitts beruhen auf folgenden Quellen: (The Transport Layer Security (TLS) Protocol Version 1.2, 2008)

Das TLS Protokoll bietet Kommunikationsprivatsphäre über das Internet. Das Protokoll erlaubt es Client/Server Applikationen zu kommunizieren so dass sie vor Abhören, Manipulation und Nachrichtenfälschung geschützt sind.
Das primäre Ziel von TLS ist es Privatsphäre und Datenintegrität zwischen zwei kommunizierenden Applikationen sicherzustellen. Das Protokoll besteht aus zwei Schichten. TLS Record Protocol und TLS Handshake Protocol.

Das TLS Record Protocol stellt Verbindungssicherheit sicher und hat zwei grundlegende Eigenschaften:

- Die Verbindung ist privat. Symmetrische Kryptographie wird für Datenverschlüsselung benutzt (z.B. DES, RC4, etc.). Die Schüssel für diese symmetrische Verschlüsselung werden für jede Verbindung generiert und basieren auf einem Geheimnis (Secret) welches vom TLS Handshake Protocol ausgehandelt wurde. Das Record Protocol kann auch ohne Verschlüsselung genutzt werden.

- Die Verbindung ist verlässlich (reliable). Nachrichtentransport beinhaltet ein Message Integritäts-Check mit einem keyed MAC. Sichere Hashfunktionen (z.B. SHA, MD5, etc.) werden für die MAC Berechnung verwendet.

Das Record Protocol wird benutzt, um diverse Protokolle auf höheren Schichten zu kapseln. Das TLS Handshake Protocol erlaubt es Client und Server sich gegenseitig zu authentifizieren und Verschlüsselungsalgorithmen sowie Kryptographische Schüssel auszuhandeln, bevor das Applikationsprotokoll Daten übermittelt oder empfängt.

Das TLS Handshake Protocol stellt Verbindungssicherheit sicher und hat drei grundlegende Eigenschaften:

- Die Identität eines Peers kann mit asymmetrischen, oder public key Kryptographie (z.B. RSA, DSS, etc.) authentifiziert werden. Diese Authentifizierung kann optional gemacht werden, ist aber normalerweise für mindestens einen Peer erforderlich.
- Die Aushandlung eines geteilten Geheimnisses (shared Secret) ist sicher. Das Secret kann nicht abgehört werden. Für jede authentifizierte Verbindung kann das Geheimnis nicht erlangt werden, auch nicht durch Angreifer, die in der Mitte der Verbindung sitzen.
- Die Verhandlung ist verlässlich. Kein Angreifer kann die Verhandlungskommunikation manipulieren ohne dass es die Kommunikationspartner bemerken würden.

4.8 **CredSSP**

Das CredSSP-Protokoll gibt einer Applikation die Möglichkeit, sicher die Anmeldedaten eines Benutzers vom Client an den Zielserver zu senden. Es ist ein zusammengesetztes Protokoll, welches sich auf andere standardbasierte Sicherheitsprotokolle stützt. Es verwendet das Transport Layer Security (TLS)-Protokoll, um einen verschlüsselten Kanal zwischen dem CredSSP-Client und CredSSP-Server aufzubauen und alle weiteren Nachrichten werden über den TLS-Kanal gesendet. Zusätzlich wird der Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO) für die Authentifizierung von Client und Server in der verschlüsselten TLS-Sitzung verwendet. (Microsoft, 2021)

4.9 **GSS-API**

Die GSS-API (Generic Security Service Application Program Interface) ist eine Programmierschnittstelle für Anwendungen, die auf Sicherheitsdienste zugreifen. Die API ermöglicht es den Providern von Sicherheitsdiensten GSSAPI-kompatible Implementierungen von ihren Sicherheitsdiensten anzubieten. Die GSS-API wird in den meisten Fällen von den Kommunikationsprotokollen selbst aufgerufen, um die Kommunikation mit Authentifizierungs-, Integritäts- und/oder Vertraulichkeits-Sicherheitsdiensten zu schützen. Das entscheidende Feature von GSSAPI-Anwendungen ist der Austausch von sogenannten Tokens (undurchsichtigen Nachrichten), die die Implementierungsdetails vor der übergeordneten Anwendung verbergen. Dies ermöglicht es der Client- und der Serverseite der Anwendung, eine bestimmte Anzahl von Token auszutauschen, um einen Sicherheitskontext zwischen ihnen herzustellen. (Wikipedia, 2021; Generic Security Service Application Program Interface Version 2, Update 1, 2000)

4.10 **SPNEGO**

SPNEGO (Simple and Protected GSSAPI Negotiation Mechanism) ist ein Mechanismus der GSS-API. Dieser Mechanismus wird von Client- sowie Server-Anwendungen verwendet, um die Wahl der Sicherheitsdienste auszuhandeln. Daher kommt SPNEGO zum Einsatz, wenn eine Client-Anwendung sich bei einem Remote-Server authentifizieren möchte, aber keiner der beiden Kommunikationsteilnehmer weiss, welche Authentifizierungsprotokolle das Gegenüber unterstützt. SPNEGO-Protokoll ermittelt in diesem Fall, welche GSSAPI-Sicherheitsdienste verfügbar sind. Anschliessend wird einer von diesen ausgewählt, mit dem dann alle weiteren Sicherheitsoperationen fortgesetzt werden. (Wikipedia, 2020; MS-SPNG, 2021)

4.11 **NTLM**

NTLM (NT LAN-Manager) ist ein Authentifizierungsverfahren, welches von dem Softwarehersteller Microsoft entwickelt wurde. Anfänglich wurde das Protokoll fast ausschliesslich in Produkten von Microsoft implementiert. Mittlerweile wird NTLM aber auch von Systemen verwendet, die nicht von Microsoft stammen. Das NTLM-Protokoll ermöglicht die gegenseitige Authentifizierung zwischen verschiedenen Computern und Servern. Das Protokoll ist so aufgebaut, dass sich ein Client mit seinem Benutzernamen und dem dazugehörigen Passwort authentifiziert. Dabei werden die zur Authentifizierung notwendigen Informationen zwischen dem Gerät des Benutzers und einem Server ausgetauscht. Da der Server Informationen über alle Benutzer hat, kann er den Zugang prüfen und diesen dann freigeben oder ablehnen. (Wikipedia, 2021; IONOS, 2020)

4.12 **Kerberos**

Kerberos ist ein verteilter Authentifizierungsdienst für offene Computernetze. Kerberos basiert auf dem Needham-Schroeder-Protokoll zur Authentifizierung. Die eigentliche Authentifizierung übernimmt eine vertrauenswürdige dritte Partei. Clients erhalten verschlüsselte Tickets von dieser dritten Partei, mit welchen sie sich dann gegenüber verschiedenen Diensten authentifizieren können. (Kryptowissen.de, 2016)

4.13 **Fazit**

Ziel dieses Kapitels war es, durch die Betrachtung der Theorie die einzelnen relevanten Themen und deren Zusammenhänge zu verstehen. In der folgenden Abbildung (Abbildung 3) werden die betrachteten Begriffe in einem Gesamtbild mit ihren Zusammenhängen dargestellt.

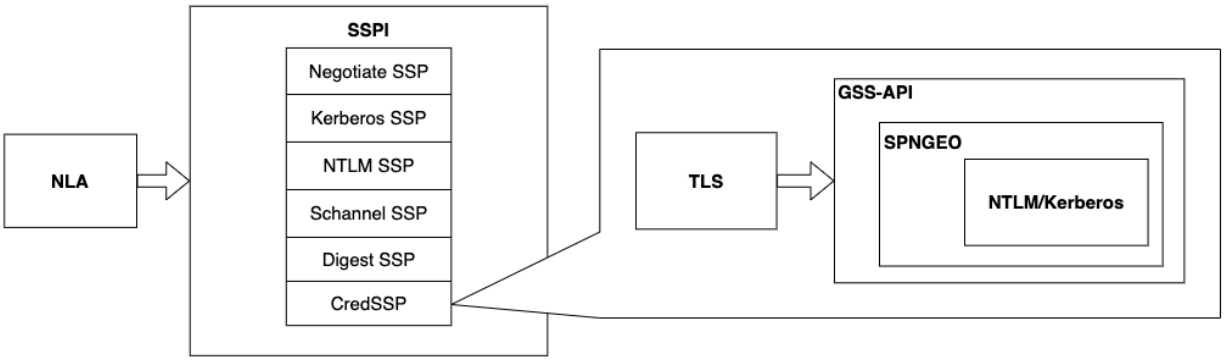


Abbildung 3: Gesamtbild der Theorie (Eigene Darstellung)

Die wichtigsten Erkenntnisse werden im Folgenden zusammengefasst:

- NLA ist eine Windows Funktion, welche die Client Authentifizierung auf Netzwerkeben ermöglicht.
- SSPI wird von NLA aufgerufen, um den SSP auszuwählen, welcher die Authentifizierung durchführt. (RDP nutzt den SSP «CredSSP»)
- CredSSP ist ein zusammengesetztes Protokoll, welches aus TLS und NTLM/Kerberos besteht.
- TLS ist ein Kommunikationsprotokoll, welches von CredSSP verwendet wird. Es erzeugt einen verschlüsselten Kanal.
- GSS-API ist eine Programmierschnittstelle für Anwendungen, die auf Sicherheitsdienste zugreifen.
- SPNEGO ist ein GSS-API-Mechanismus, um die Wahl der Sicherheitsdienste auszuhandeln.
- NTLM & Kerberos sind zwei Sicherheitsdienste, die verschiedene Authentifizierungsverfahren anbieten.

5 Analyse: Machbarkeit von Man-in-the-Middle

In diesem Kapitel wird die Hauptaufgabe unserer Arbeit angegangen. Hier wird eine mit NLA/CredSSP geschützte RDP-Verbindung auf die Durchführbarkeit eines Man-in-the-Middle-Angriffes analysiert. Zunächst wird das RDP-Protokoll selbst untersucht. Anschliessend werden die in einer NLA/CredSSP-geschützten RDP-Verbindung zum Einsatz kommenden Protokolle nacheinander untersucht. Anhand dieser theoretischen Analysen soll die Durchführbarkeit des Angriffes beurteilt werden können. Das Ergebnis dieser Analyse ist die Voraussetzung, um den Man-in-the-Middle-Proof of Concept zu implementieren.

5.1 **RDP**

Als erstes wird das RDP-Protokoll selbst genauer untersucht. Obwohl es sich bei diesem MitM-Versuch um eine NLA/CredSSP-geschützte Verbindung handelt, wird der Angriff auf RDP selbst ausgeführt. Daher ist es wichtig, sich das entsprechende Wissen über RDP anzueignen, welches für diese Machbarkeitsanalyse relevant sein könnten.

Der Inhalt und die Ausführungen des folgenden Abschnitts beruhen sich auf folgenden Quellen: (Reiner, Shaked, 2020; Microsoft, 2021)

5.1.1 Verbindungsaufbau

In diesem Abschnitt wird der RDP-Verbindungsaufbau bzw. die RDP-Verbindungssequenz genauer betrachtet. Ziel der RDP-Verbindungssequenz ist es, Client- und Servereinstellungen auszutauschen und gemeinsame Einstellungen festzulegen, die für die Dauer der RDP-Verbindung verwendet werden sollen. Anhand dieser Einstellungen können Eingaben, Grafiken und andere Daten zwischen Client und Server ausgetauscht und verarbeitet werden.

Die folgende Abbildung (Abbildung 4) zeigt den kompletten Ablauf einer RDP-Verbindungsaufbau. Die einzelnen, nicht optionalen Schritte werden im Folgenden kurz betrachtet und daraufhin bewertet, ob sie für die Machbarkeitsanalyse relevant sind.

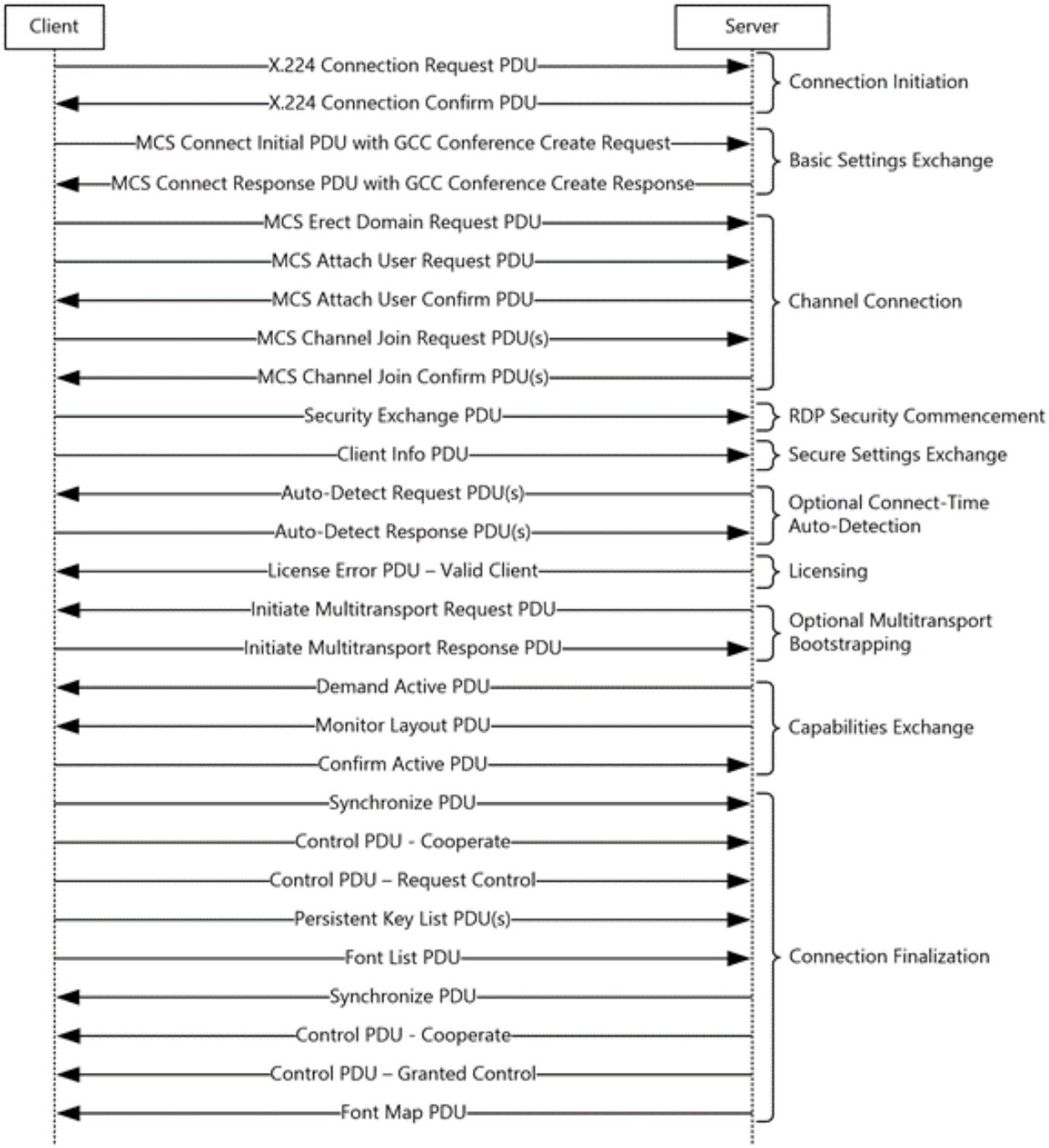


Abbildung 4: RDP Connection Sequence (Quelle: (Microsoft, 2021))

Connection Initiation

Abbildung 5: RDP Connection Initiation (Quelle: (Microsoft, 2021))

Dieses Paket enthält einen RDP Negotiation Request, der einige Verbindungsflags und die vom Client unterstützten Sicherheitsprotokolle enthält. Die Verbindung wird vom Server mit einer RDP Negotiation Response bestätigt, und wird auch verwendet, um den Client über das

ausgewählte Sicherheitsprotokoll zu informieren, das während der gesamten Verbindungsdauer verwendet wird (Abbildung 5).

Für unsere MitM-Analyse bedeutet das, dass der Server in dieser ersten Phase des Verbindungsaufbaus dem Client mitteilt, dass er eine Authentifizierung über NLA erzwingt und dass dafür das Sicherheitsprotokoll CredSSP verwendet werden soll.

Basic Settings Exchange

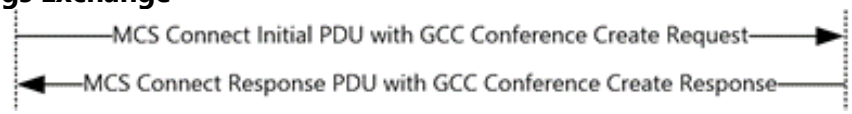


Abbildung 6: RDP Basic Settings Exchange (Quelle: (Microsoft, 2021))

In diesen Paketen werden die Grundeinstellungen zwischen dem Client und dem Server ausgetauscht (Abbildung 6). Diese Einstellungen umfassen drei Arten:

1. Core Daten: RDP-Version, Desktop-Auflösung, Farbtiefe, Tastaturinformationen, Hostname usw.
2. Security Daten: Verschlüsselungsmethoden, Länge der Sitzungsschlüssel, Server Zertifikat, Server Random
3. Network Daten: Die vom Client angefragten virtuellen Kanäle. Der Client wird darauf vom Server mit einer Liste von virtuellen Kanälen, die in der RDP-Sitzung verwendet werden sollen, informiert.

Wenn der MitM den Verschlüsselungsschlüssel aus der CredSSP-Authentifizierung auslesen oder selbst neu berechnen kann, sollten die in dieser Phase gesendeten Nachrichten keine Probleme verursachen. Der MitM sollte diese Nachrichten ohne Probleme weiterleiten können. Sobald Enhanced Security verwendet wird, wird kein symmetrischer Schlüssel zwischen den beiden Kommunikationsteilnehmern erzeugt. Daher sind auch die unter Security Data gesendeten Daten nicht relevant.

Channel Connection

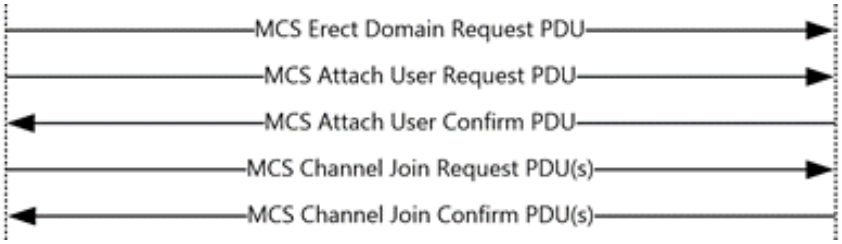


Abbildung 7: RDP Channel Connection (Quelle: (Microsoft, 2021))

In dieser Phase stellt der Client eine Verbindung zu jedem Kanal aus der Liste her, welche er zuvor vom Server erhalten hat. Sobald mit allen Kanälen eine Verbindung aufgebaut wurde, werden die folgenden Daten in die jeweiligen virtuellen Kanäle umgeleitet (Abbildung 7).

Auch sollte es möglich sein, diese Nachrichten, die für die Herstellung der einzelnen Kanäle verantwortlich sind, weiterzuleiten. Was bei der bisherigen Analyse nicht ausgeschlossen werden kann, ist, ob der MitM aufgrund der Weiterleitung auch einen RDP-Dienst anbieten muss, damit er die zukünftigen Nachrichten über einen bestimmten Kanal tatsächlich verfolgen kann. Dies sollte aber kein Problem darstellen, da man den RDP-Dienst auch auf dem MitM-Rechner aktivieren könnte.

Security Commencement

Abbildung 8: RDP Security Exchange PDU (Quelle: (Microsoft, 2021))

Hier schickt der Client dem Server das Client-Random, welches er mit dem öffentlichen Schlüssel des Servers verschlüsselt (Abbildung 8). Diesen hat er zuvor im Schritt «Basic Settings Exchange» vom Server erhalten. Der Client und der Server verwenden dann die beiden 32-Byte Randoms, um einen symmetrischen Session Encryption Key zu erstellen. Von diesem Punkt an werden alle folgenden RDP-Nachrichten mit dem Verschlüsselungsalgorithmus RC4 verschlüsselt.

Dieser Schritt wird bei Verwendung von Enhanced Security automatisch vom RDP-Protokoll übersprungen und ist daher für unsere Machbarkeitsanalyse nicht relevant.

Secure Settings Exchange

Abbildung 9: RDP Secure Settings Exchange (Quelle: (Microsoft, 2021))

In dieser Phase sendet der Client dem Server vertrauliche Client-Daten (z. B. Benutzername, Kennwort und Auto-Reconnect-Cookie) (Abbildung 9).

Wie beim Schritt «Basic Setting Exchange» sollte auch diese Nachricht vom MitM problemlos weitergeleitet werden können. Durch die Kenntnis des Verschlüsselungsschlüssels sollte der MitM auch in der Lage sein, diese Nachricht zu entschlüsseln und die notwendigen Daten auszulesen. Diese muss er dann wieder verschlüsseln und an den Server weiterleiten.

Licensing

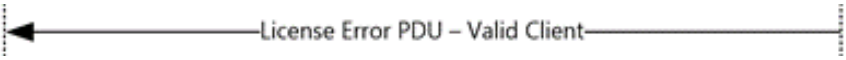


Abbildung 10: RDP Licensing (Quelle: (Microsoft, 2021))

Hier überträgt der Server eine Lizenz zum Client (Abbildung 10). Der Client speichert diese Lizenz und sendet bei nachfolgenden Verbindungen die Lizenz zur Validierung an den Server. In den meisten Fällen ist aber kein Lizenzierungsserver für den RDP-Server konfiguriert. In diesem Fall genehmigt der RDP-Server einfach die Lizenz des Clients.

Der MitM sollte auch in der Lage sein, diese Nachricht vom Server an den Client weiterzuleiten. Da der Server die Lizenz des Clients genehmigt, wenn er nicht über einen eigenen Lizenzierungsserver verfügt, sollte auch dies einen MitM-Angriff nicht verhindern.

Capabilities Exchange

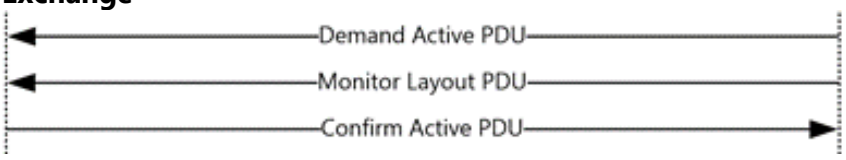


Abbildung 11: Capabilities Exchange (Quelle: (Microsoft, 2021))

In dieser Phase des Verbindungsaufbaus sendet der Server die von ihm unterstützten Features an den Client. Der Client antwortet auf diese Nachricht mit einer Bestätigung, die wiederum seine eigenen Features beinhaltet. (Abbildung 11)

Diese Nachrichten, die die unterstützten Features des jeweils anderen ankündigen, sollten einfach weitergeleitet werden können. Der MitM in seiner einfachsten Funktion muss lediglich in der Lage sein, die Kommunikation abzuhören und keine eigenen Features erzwingen. Daher müssen an diesen Nachrichten auch keine Änderungen vorgenommen werden.

Connection Finalization

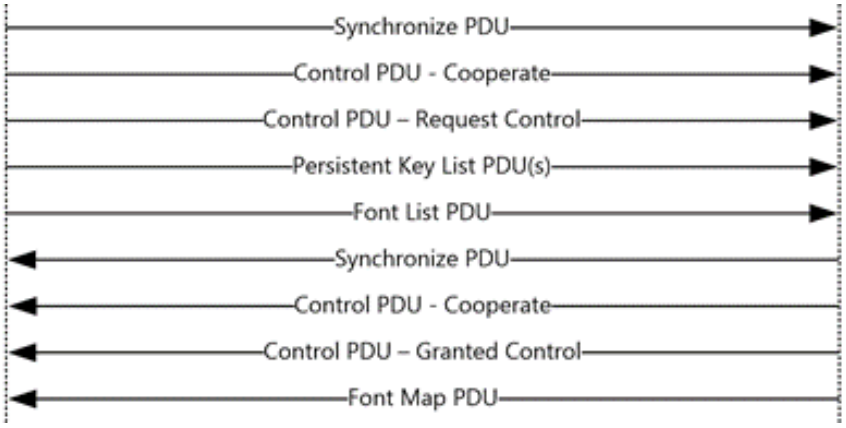


Abbildung 12: Connection Finalization (Quelle: (Microsoft, 2021))

Zum Schluss werden zwischen Client und Server die notwendigen PDUs, Nutzdaten und Verwaltungsinformationen der Schichten (OSI-Schichtenmodell), ausgetauscht, um den Verbindungsaufbau abzuschliessen (Abbildung 12). Die in dieser Phase gesendeten Client-zu-Server-PDUs haben keine Abhängigkeiten von den Server-zu-Client-PDUs.

Wie im Capabilities Exchange ist es auch hier möglich, diese letzten Nachrichten des Verbindungsaufbaus ohne weitere Änderungen weiterzuleiten.

5.1.2 Sicherheit

Für das RDP-Protokoll gibt es zwei Arten von Sicherheitsstufen: «Standard Security» oder «Enhanced Security».

5.1.2.1 Standard Security

In der Standard-Security wird der Datenverkehr mit dem Verschlüsselungsalgorithmus RC4 verschlüsselt. Dazu werden Zufallswerte (Randoms) von Client und Server verwendet, die beim Verbindungsaufbau von RDP (5.1.1 Verbindungsaufbau) in der Phase «Basic Settings Exchange» und «Security Commencement» ausgetauscht werden, um einen gemeinsamen symmetrischen Schlüssel zu erzeugen.

5.1.2.2 Enhanced Security

Bei dieser Art von Sicherheit lagert RDP alle Sicherheitsoperationen, wie zum Beispiel Ver- und Entschlüsselung, Integritätsprüfungen etc., an eines der folgenden externen Sicherheitsprotokolle aus:

- TLS
- CredSSP (TLS + NTLM/Kerberos)
- RDSTLS – (Standard RDP Security erweitert mit TLS)

Wenn Enhanced Security verwendet wird, wird in der ersten Phase des RDP-Verbindungsaufbaus («Connection Initiation») die Option «Enhanced Security» gewählt. Das

bedeutet, dass die Phase «Security Commencement» (Generierung von symmetrischen Schlüsseln) für den Verbindungsaufbau nicht erforderlich ist und daher ausgelassen wird.

Einer der Hauptvorteile der Enhanced Security ist es, dass es die Authentifizierung auf Netzwerkebene (NLA) ermöglicht. Die Auswahl des Sicherheitsprotokolls kann entweder «negotiation-based» oder «direct» erfolgen.

Negotiation-based
Client und Server wählen nach der Initialisierung der RDP-Verbindung (Phase «Connection Initiation») ein externes Sicherheitsprotokoll aus. Für dieses führen sie dann den Handshake durch. Ab dann werden alle anderen Phasen der RDP-Verbindung innerhalb dieses externen Sicherheitsprotokolls gekapselt.

Direct
Diese Option bevorzugt die Sicherheit gegenüber der Kompatibilität. Hier beginnt der Client nach der Phase «Connection Initiation» direkt mit dem Handshake des externen Sicherheitsprotokolls, bevor er irgendwelche RDP-bezogenen Daten sendet.

5.1.3 Fazit

- CredSSP-geschütztes RDP gehört zur Stufe «Enhanced Security».
- Bei Verwendung von Enhanced Security entfällt der Schritt «Security Commencement» im Verbindungsaufbau.
- CredSSP wird bei RDP durch Erzwingen der NLA-Authentifizierung automatisch ausgewählt.
- Im ersten Schritt «Connection Initiation» des Verbindungsaufbaus informiert der Server den Client über die Erzwingung der NLA.
- Bei Verwendung von NLA wird nach «Connection Initiation» zuerst die Authentifizierung mittels CredSSP durchgeführt und erst dann die RDP-Verbindung aufgebaut.
- Alle Sicherheitsoperationen (Authentifizierung, Ver-/Entschlüsselung, Integritätsprüfungen, etc.), werden an CredSSP ausgelagert.
- Alle weiteren Schritte des Verbindungsaufbaus sollten in einem MitM-Fall einfach weitergereicht werden. (Voraussetzung: Verschlüsselungsschlüssel bekannt)
- Wie in diesem Abschnitt «RDP» festgestellt, wird, wenn die Authentifizierung durch NLA (explizit) geschützt ist, diese unmittelbar nach dem ersten Schritt des RDP-Verbindungsaufbaus durchgeführt, bevor die anderen Schritte des Verbindungsaufbaus ausgeführt werden. In diesem Fall entfällt auch der Schritt «Security Commencement» des RDP-Verbindungsaufbaus, bei dem die Schlüssel für die Verschlüsselung in der Standard-Security erzeugt werden. Das bedeutet, dass die RDP-Verbindung in der Regel nur in irgendeiner Weise durch CredSSP oder TLS verschlüsselt wird. Da CredSSP ein Sicherheitsdienst ist, der auf TLS aufbaut, ist es notwendig, dass in einem ersten Schritt das TLS-Protokoll selbst untersucht wird, bevor die Analyse auf CredSSP fortgesetzt wird.

5.2 **TLS**

In diesem Kapitel wird das Transport Layer Security Protokoll (TLS) genauer analysiert. TLS bildet die Grundlage für die Enhanced Security RDP Verbindung, da TLS den sicheren Kanal bereitstellt, über welchen die gesamte Kommunikation läuft.

Der Inhalt und die Ausführungen des folgenden Abschnitts beruhen sich auf folgenden Quellen: (The Transport Layer Security (TLS) Protocol Version 1.2, 2008)

5.2.1 Funktionsweise

Die kryptographischen Parameter der Session werden vom TLS Handshake (5.2.2 Handshake) Protokoll produziert.
Wenn ein Client und ein Server beginnen zu kommunizieren, einigen sie sich auf Protokollversion, wählen kryptographische Algorithmen, authentifizieren sich gegenseitig (optional) und benutzen public-key Verschlüsselungstechniken um ein geteiltes Geheimnis (shared secret) zu generieren.

Folgende Schritte beinhaltet das TLS Handshake Protocol:

- Hello Messages austauschen um sich auf Algorithmen zu einigen, Random Values auszutauschen, und auf Session Wiederaufnahme (Session Resumption) zu prüfen.
- Die nötigen kryptographischen Parameter austauschen, um dem Client und Server zu erlauben sich auf ein Premaster Secret zu einigen.
- Zertifikate und kryptographische Informationen austauschen, um Client und Server zu erlauben sich zu authentifizieren.
- Master Secret aus Premaster Secret und aus den ausgetauschten Random Values generieren.
- Security Parameter bereitstellen für den Record Layer
- Client und Server erlauben zu überprüfen, ob ihr Peer die gleichen Security Parameter berechnet haben und das der Handshake ohne Manipulation eines Angreifers vonstattenging.

Höhere Schichten sollten sich nicht zu sehr darauf verlassen, dass TLS immer die stärkste mögliche Verbindung zwischen zwei Peers herstellt. Es gibt viele Angriffe wie ein Man-in-the Middle-Angreifer versuchen kann, dass zwei Entities zu der am wenigsten sicheren Methode übergehen.

5.2.2 Handshake

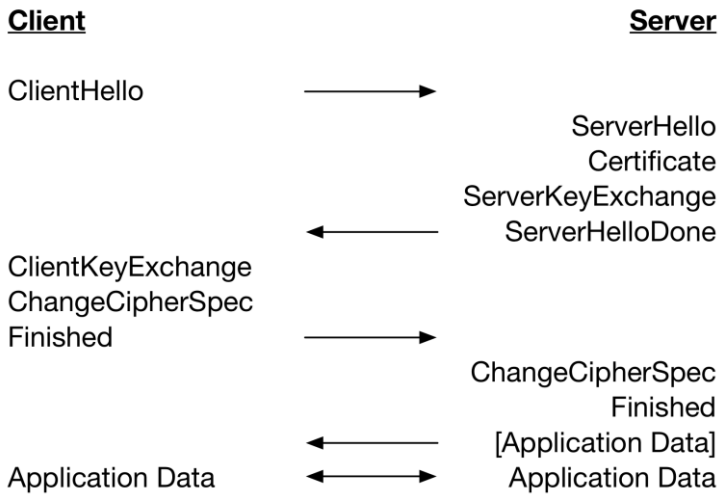


Abbildung 13: TLS Handshake (Quelle: (wolfSSL, 2021))

Der Client sendet sein ClientHello an den Server und der Server muss dann mit einem ServerHello antworten oder ein «fatal Error» tritt auf und die Verbindung schlägt fehl. Das ClientHello (Schritt 1 in der Abbildung 13) und das ServerHello (Schritt 2) werden verwendet, um die Fähigkeit für Sicherheitsverbesserungen einzurichten. ClientHello und ServerHello richten folgende Attribute ein: Protokollversion, Session ID, Cipher Suite, Kompressionsmethode (Compression Method). Zusätzlich werden zwei zufällige Werte generiert und ausgetauscht.

Der effektive Schüsselaustausch benötigt bis zu vier Messages: Server Certificate, Server Key Exchange, Client Certificate und Client Key Exchange. Die Key Exchange Methoden tauschen Secrets aus im Bereich von 46 Bytes Länge aufwärts.

Nach den Hello Messages sendet der Server sein Zertifikat an den Client, wenn er authentifiziert werden muss. Zusätzlich kann eine Server Key Exchange Message (Schritt 2) gesendet werden, wenn es notwendig ist. (z.B. wenn der Server kein Zertifikat hat oder die benötigten Parameter für die Schlüsselgenerierung nicht aus dem Zertifikat entnommen werden). Wenn der Server dann authentifiziert ist, verlangt er möglicherweise ein Zertifikat von Client, wenn das gemäss der gewählten Cipher Suite angemessen ist. Dann sendet der Server eine Hello Server done Message (Schritt 2), mit welcher er signalisiert das die hello-message Phase des Handshakes abgeschlossen ist. Der Server wartet dann auf eine Antwort des Clients.

Wenn der Server eine Certificate Request Message gesendet hat, muss der Client eine Certificate Message senden. Dann wird der Client Key Exchange (Schritt 3) gesendet und der Inhalt dieser Message hängt vom gewählten Public-Key Algorithmus ab, der zwischen ClientHello und ServerHello gewählt wurde.

Jetzt sendet der Client eine Change Cipher Spec Message (Schritt 3) und der Client kopiert die ausstehende Cipher Spec in die aktuelle Cipher Spec. (D.h. dass der Client jetzt auf die ausgehandelte Cipher Suite wechselt). Und unmittelbar darauf sendet der Client die finished Message.

Der Server sendet ebenfalls eine Change Cipher Spec Message und eine Finished Message (Schritt 4). Jetzt ist der Handshake komplett und Client/Server können beginnen Applikationsdaten auszutauschen. Applikationsdaten dürfen nicht vor dem Beenden des ersten Handshakes gesendet werden (bevor eine Cipher Suite etabliert wurde, ausser TLS_NULL_WITH_NULL_NULL).

5.2.3 Berechnung Premaster und Master Secret

Mit der Server Key Exchange Message überträgt der Server entweder einen RSA public Key um das Premaster Secret zu verschlüsseln oder er überträgt einen Diffie-Hellman public Key mit welchem der Client dann den Key Exchange komplettieren kann. Danach sendet dann der Client seine Client Key Exchange Message, mit welcher das Premaster Secret gesetzt wird, entweder durch direkte Übermittlung eines mit RSA verschlüsselten Secrets oder durch die Übertragung von Diffie-Hellman Parametern, was beiden Seiten erlaubt das gleiche Premaster Secret zu berechnen.
Falls das Premaster Secret mit RSA verschlüsselt übertragen wird, generiert der Client das Premaster Secret.
Master Secret = PRF(pre_master_secret, «master secret», ClientHello.random + ServerHello.random). Das Master Secret ist immer exakt 48 Byte lang. Die Länge des Premaster Secret kann je nach Algorithmus variieren. Das Premaster Secret sollte aus dem Speicher gelöscht werden, sobald das Master Secret berechnet wurde. Von diesem Master Secret werden alle verwendeten Schlüssel für die Verschlüsselung und für die MACs abgeleitet. Wie genau die Schlüssel abgeleitet werden, hängt vom verwendeten Algorithmus ab.

Nach erfolgreichem Ende des Handshakes besteht nun ein sicherer Kanal zwischen zwei Kommunikationspartnern. Dieser Kanal ist die Grundlage für das CredSSP Protokoll, welches im nächsten Kapitel genauer analysiert wird.

5.3 **CredSSP**

In diesem Abschnitt wird das CredSSP Protokoll genauer analysiert, um die Anmeldedaten eines Benutzers von einem Client an einem Server zu senden. Bevor die RDP-Verbindung zum Server hergestellt wird, muss sich der Benutzer mit dem CredSSP Protokoll authentifizieren. Das Aktivieren der Windows-NLA-Funktion auf der Serverseite erzwingt das CredSSP-Protokoll.

Der Inhalt und die Ausführungen des folgenden Abschnitts beruhen sich auf folgenden Quellen: (Microsoft, 2021)

5.3.1 Funktionsweise

Für die Funktionsweise des Protokolls sind einige Informationen beim Client und Server vorausgesetzt, damit sich der Client erfolgreich authentifizieren kann.

CredSSP-Client	CredSSP-Server
• Hat Zugriff auf die Anmeldedaten des Benutzers • Für die Nonce Generierung muss eine Quelle kryptografischer nützlicher Zufallszahlen verfügbar sein • GSS-kompatibles Authentifizierungsprotokoll muss vorhanden sein für die gegenseitige Client/Server-Authentifizierung (SPNEGO)	• Für die Nonce Generierung muss eine Quelle kryptografischer nützlicher Zufallszahlen verfügbar sein • Verfügt über ein x.509-Zertifikat für die Verwendung von TLS (Das Zertifikat kann selbstsigniert oder von einer fremden Zertifizierungsstelle ausgestellt sein) • GSS-kompatibles Authentifizierungsprotokoll muss vorhanden sein für die gegenseitige Client/Server-Authentifizierung (SPNEGO)

Tabelle 1: CredSSP Funktionsweise

Anhand dieses Sequenzdiagramms (Abbildung 14) werden hier die einzelnen Schritte detaillierter beschrieben.

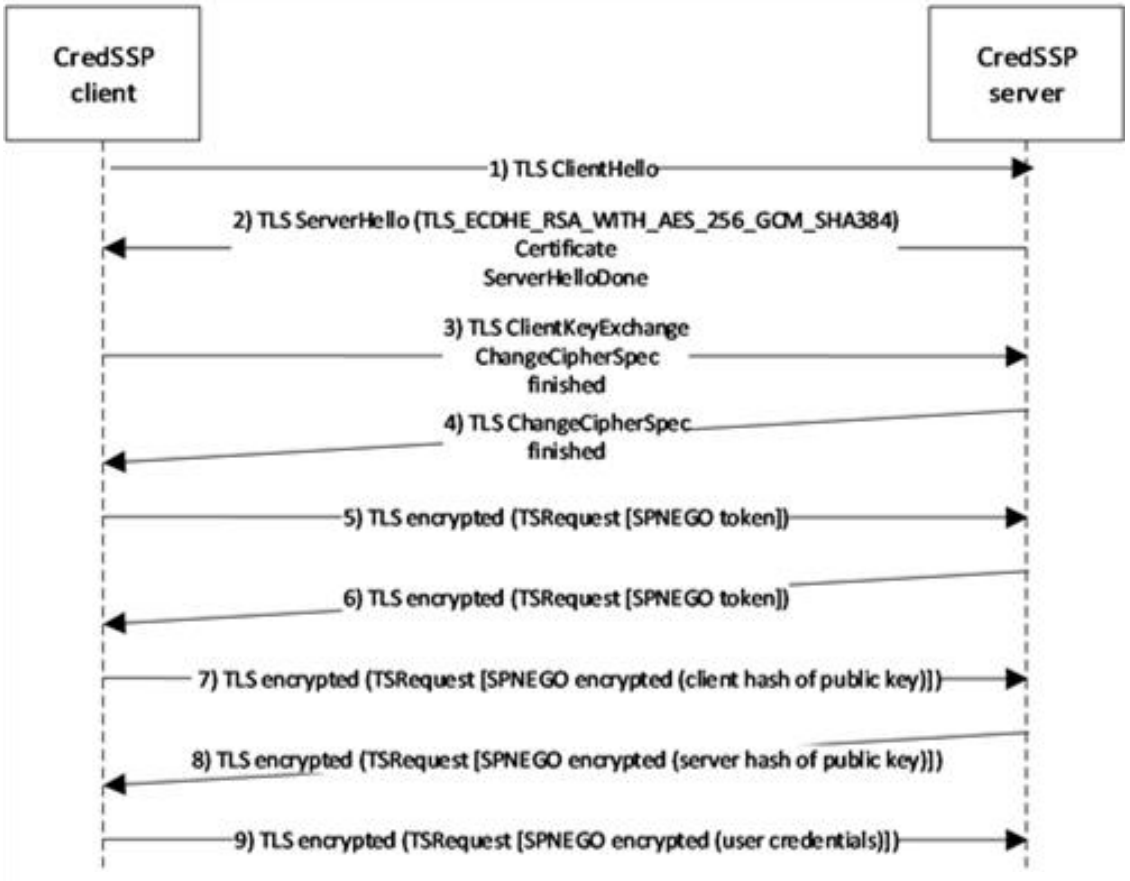


Abbildung 14: CredSSP Handshake (Quelle: (Microsoft, 2021))

Schritte 1-4 => Es wird ein TLS-Handshake (5.2.2 Handshake) zwischen CredSSP-Client und CredSSP-Server durchgeführt. Hier werden die Cipher Suite, Zertifikate und zufällig generierte Werte ausgetauscht. Nach dem TLS Handshake werden alle CredSSP-Messages über den verschlüsselten TLS-Kanal gesendet.

Schritte 5-6 => Hier verwendet das CredSSP-Protokoll das SPNEGO-Protokoll für die Server- und Benutzerauthentifizierung und kann aus einer Reihe möglicher Authentifizierungsmechanismen wie z.B. NLTM oder Kerberos wählen. Im verschlüsselten TLS-Kanal wird der SPNEGO-Handshake zwischen Client und Server für die gegenseitige Authentifizierung abgeschlossen und es wird ein Chiffrierschlüssel erzeugt.

Schritte 7-8 => Für die Bindung an die TLS-Sitzung wird der erzeugte SPNEGO Chiffrierschlüssel verwendet. Der Client verschlüsselt den Hash des öffentlichen Schlüssels aus dem X.509 Zertifikat des Servers mit dem SPNEGO-Chiffrierschlüssel und sendet diesen verschlüsselten Hash an den Server. Der Server verifiziert den öffentlichen Schlüssel, ob der im TLS-Handshake verwendet wurde und sendet eine verschlüsselte Bestätigung mit dem SPNEGO-Chiffrierschlüssel an den Client.

Schritt 9 => Am Ende des CredSSP-Handshakes werden dann die verschlüsselten Benutzeranmeldeinformationen mit dem SPNEGO-Chiffrierschlüssel und vom TLS-Kanal geschützt an den Server gesendet.

Alle weiteren Daten werden verschlüsselt unter TLS gesendet.

Da CredSSP eine Kombination aus TLS und einem Authentifizierungsmechanismus ist, wird nachfolgend zuerst die Auswahl des Authentifizierungsmechanismus mit SPNEGO genauer analysiert und danach die zwei verfügbaren Authentifizierungsmechanismen NTLM und Kerberos.

5.4 **SPNEGO (GSS-API)**

Wie in der Theorie beschrieben, ist das SPNEGO-Protokoll ein Erweiterungsmechanismus der GSS-API, das ermittelt, welche GSS-API-Mechanismen, NTLM oder Kerberos, zur Authentifizierung des Clients verwendet werden können. Das Protokoll wählt dann den sichersten Mechanismus von den verfügbaren aus und leitet dann alle weiteren Sicherheitsoperationen an diesen Mechanismus weiter. Wie die GSS-API bietet auch SPNEGO selbst keine Sicherheit, sondern wird nur verwendet, um zu bestimmen, welcher Sicherheitsmechanismus verwendet werden soll.

5.5 **NTLM**

Bei der Analyse des CredSSP-Handshakes (5.3 CredSSP) zeigte sich, dass im Rahmen des Handshakes in den Schritten 7 und 8 (Abbildung 14) der öffentliche Schlüssel des Servers zusammen mit einer konstanten Zeichenkette und einem Zufallswert verkettet und mit dem SPNEGO-Token verschlüsselt über den aufgebauten TLS-Kanal zwischen den Parteien ausgetauscht wird. Dieser wird dann von der empfangenden Partei geprüft und soll einen Man-in-the-Middle-Angriff verhindern.

Durch die Analyse des SPNEGO-Protokolls konnte festgestellt werden, dass es sich bei den SPNEGO-Tokens entweder um NTLM-Authentifizierung oder Kerberos-Authentifizierung handelt. Daher wird in einem ersten Schritt das Authentifizierungsverfahren mittels NTLM untersucht und dabei die bei diesem Verfahren ausgetauschten Nachrichten genauer betrachtet. Ziel dieser Analyse ist es, herauszufinden, wie der im CredSSP-Handshake von SPNEGO verwendete Verschlüsselungsschlüssel erzeugt wird.

Der Inhalt und die Ausführungen des folgenden Abschnitts beruhen sich auf folgenden Quellen: (Glass, 2006; Microsoft, 2021)

5.5.1 Messages

Das NTLM-Authentifizierungsprotokoll besteht aus drei Nachrichtentypen (Abbildung 15), NEGOTIATE_MESSAGE, CHALLENGE_MESSAGE und AUTHENTICATE_MESSAGE, die während der Authentifizierung verwendet werden. Diese NTLM-Nachrichten werden zwischen dem Client und dem Server ausgetauscht. Diese Nachrichten werden in das Anwendungsprotokoll eingebettet, das die NTLM-Authentifizierung verwendet. Das NTLM-Protokoll kommt erst zum Einsatz, wenn es vom Anwendungsprotokoll aufgerufen wird, um eine Authentifizierung durchzuführen. NTLM selbst baut keine Transportverbindungen auf und verwendet daher die bereits aufgebaute TLS-Verbindung für die Durchführung der Authentifizierung.

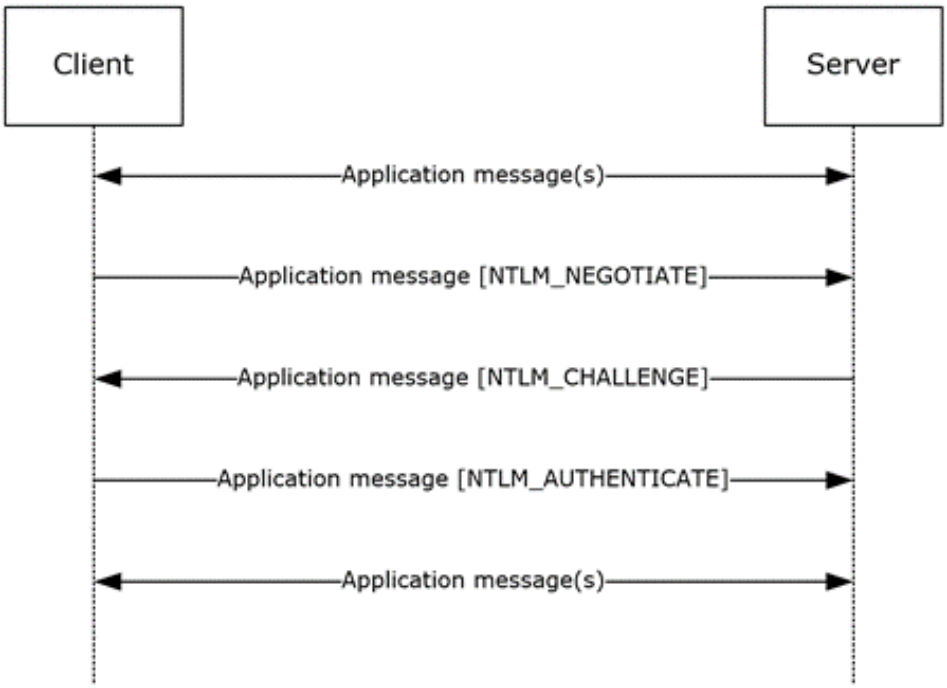


Abbildung 15: NTLM Authentication Messages (Quelle: (Microsoft, 2021))

5.5.1.1 NEGOTIATE_MESSAGE

Die NTLM-Type 1-Nachricht wird vom Client an den Server gesendet, um die NTLM-Authentifizierung zu initiieren. Ihr Hauptzweck besteht darin, die Grundregeln für die Authentifizierung festzulegen, indem sie die unterstützten Optionen über die Flags mitteilt. Optional kann sie dem Server auch den Namen der Arbeitsstation des Clients und die Domäne

mitteilen, in der die Client-Arbeitsstation Mitglied ist. Diese Informationen werden vom Server verwendet, um festzustellen, ob der Client für die lokale Authentifizierung in Frage kommt.

5.5.1.2 CHALLENGE_MESSAGE

Die NTLM-Type 2-Nachricht wird vom Server an den Client als Antwort auf die Typ-1-Nachricht des Clients gesendet. Sie führt die Aushandlung der Optionen mit dem Client zu Ende und stellt zudem eine Herausforderung («Challenge») für den Client dar. Mit diesem Challenge wird der Client aufgefordert, seine Identität zu beweisen. Die Nachricht kann ausserdem optionale Informationen über das Authentifizierungsziel enthalten. Diese Nachricht enthält somit die vereinbarten Sicherheitsaspekte, den Challenge und eine Nonce, die der Server generiert.

5.5.1.3 AUTHENTICATE_MESSAGE

Die NTLM-Type 3-Nachricht ist der letzte Schritt der Authentifizierung. Diese Nachricht enthält die Antwort des Clients auf die im Typ 2-Nachricht gesendete Challenge, die beweisen soll, dass der Client das Passwort des Kontos kennt, ohne das Passwort direkt zu übermitteln. In dieser Nachricht werden ausserdem das Authentifizierungsziel (Domänen- oder Servername), der Benutzername des zu authentifizierenden Kontos und der Name der Client-Arbeitsstation angegeben.

Der Server validiert die vom Client gesendete Challenge-Antwort, in dem er diese selbst nachberechnet. Wenn es sich dabei um einen Benutzernamen für ein lokales Konto handelt, kann er die Antwort anhand der Infos in seiner Kontodatenbank validieren. Wenn der Benutzername für ein Domänenkonto ist, validiert er die Antwort, indem er die User Authentifizierung Information an einen Domänencontroller (DC) sendet, der die Antwort validieren kann.

Wenn die Challenge und die Antwort beweisen, dass der Client das Kennwort des Kontos hat, ist die Authentifizierung erfolgreich, und das Anwendungsprotokoll wird gemäss seiner Spezifikation fortgesetzt. Wenn die Authentifizierung fehlschlägt, kann der Server den Status auf eine im Anwendungsprotokoll festgelegte Weise senden oder die Verbindung einfach beenden.

5.5.1.4 Arten von Challenge-Antworten

Der Client erstellt eine oder mehrere Antworten auf die in der Typ-2-Nachricht enthaltene Challenge und sendet diese in der Typ-3-Nachricht an den Server. Es gibt insgesamt sechs Arten von Antworten:

- LM (LAN Manager)-Response: Diese Antwort wird von den meisten älteren Clients gesendet und ist der standardmässige Antworttyp.
- NTLM-Response: Diese wird von NT-basierten Clients, inklusive Windows 2000 und XP, gesendet.
- NTLMv2-Respone: Ein neuerer Antworttyp, der in Windows NT Service Pack 4 eingeführt wurde. Dieser ersetzt die NTLM-Antwort auf Systemen, die NTLM Version 2 aktiviert haben.
- LMv2-Response: Der Ersatz für die LM-Response auf Systemen mit NTLM Version 2.

- NTLM2-Session Response: Wird verwendet, wenn die NTLM2-Sitzungssicherheit ohne NTLMv2-Authentifizierung ausgehandelt wird.
- Anonymous-Response: Wird verwendet, wenn ein anonymer Kontext eingerichtet wird. Die tatsächlichen Anmeldeinformationen werden nicht angezeigt, und es findet keine echte Authentifizierung statt.

5.5.1.5 Ableitung des Verschlüsselungsschlüssels

Durch die Betrachtung der NTLM-Nachrichten im vorherigen Kapitel (5.5.1 Messages) konnte festgestellt werden, dass bei der Authentifizierung zwei verschiedene Schlüssel erzeugt werden. Einer dieser beiden wird daher auch für die Verschlüsselung des öffentlichen Schlüssels des Servers von SPNEGO verwendet.
Um herauszufinden, wie diese Schlüssel generiert werden und welcher davon für die Verschlüsselung des öffentlichen Schlüssels des Servers verwendet wird, ist es notwendig, die Generierung der Schlüssel näher zu untersuchen. Dabei stellt sich die Frage, ob und wie sie von einer dritten Partei neu berechnet werden können.

Die zum Signieren und Verschlüsseln verwendeten Schlüssel werden als Nebenprodukt des NTLM-Authentifizierungsprozesses erstellt; zusätzlich zur Verifizierung der Identität eines Clients wird beim Authentifizierungs-Handshake ein Kontext zwischen Client und Server hergestellt, der den Schlüssel enthält, der zum Signieren und Verschlüsseln von Nachrichten zwischen den Kommunikationsteilnehmer benötigt wird.

5.5.1.5.1 NTLMv1

NTLM1 ist das ursprüngliche Signier- und Verschlüsselungsschema von NTLMSSP, das verwendet wird, wenn das Flag «Negotiate NTLM2 Key» nicht vereinbart wurde. Die Ableitung der NTLM1-Verschlüsselungsschlüssel besteht aus einem dreistufigen Prozess:

1. Master Key Negotiation (Key Exchange Key definieren)
2. Key Exchange (Random Key übermitteln)
3. Key Weakening

Master Key Negotiation

Im ersten Schritt wird ein 128-Bit-«Master Key» ausgehandelt, mit dem der letztendliche Signier- und Verschlussschlüssel abgeleitet und verschlüsselt wird. Dies wird durch das NTLM-Flag «Negotiate Lan Manager Key» gesteuert. Falls dieses Flag gesetzt ist, wird der Lan-Manager-Session Key als Master Key verwendet. Andernfalls wird der geeignete User Session Key verwendet. In beiden Fällen wird für die Berechnung (8.2.2 Keys) der Keys jedoch das Passwort des Clients verwendet.

Key Exchange

Wenn das Flag «Negotiate Key Exchange» gesetzt ist, erzeugt der Client einen neuen 128-Bit Random Key, der in Zukunft als Verschlüsselungsschlüssel verwendet werden soll. Dieser wird mit dem zuvor gewählten Key Exchange Key RC4-verschlüsselt und in der NTLM-Typ-3-Nachricht im Feld «Session Key» an den Server übermittelt. Der Server entschlüsselt dann diesen Wert, um den neuen Verschlüsselungsschlüssel (Random Key) zu erhalten.

Key Weakening

Zum Schluss wird der Verschlüsselungsschlüssel abgeschwächt, um den Exportbeschränkungen zu entsprechen. Denn NTLM1 unterstützt nur 40-Bit- und 56-Bit-Schlüssel. Wenn das NTLM-Flag «Negotiate 56» gesetzt ist, wird der 128-Bit Verschlüsselungsschlüssel (Random Key) auf 56 Bits abgeschwächt und sonst auf 40 Bits.

Dieser Schritt wird jedoch nur durchgeführt, falls der Verschlüsselungsschlüssel sich aus einem Lan-Manager-Session Key abgeleitet hat. Denn die LM- und NTLM-User-Session Keys basieren auf den Kennwort-Hash und nicht auf den Challenge Responses, in denen aber auch das Kennwort des Benutzers einfliesst. Ein bestimmtes Kennwort ergibt unter NTLM1 immer denselben User Session Key. Die Schwächung der von LM- oder NTLM-User Session Key abgeleiteten Verschlüsselungsschlüssel wurde nicht als notwendig eingestuft, da diese User Session Keys leicht aus dem Kennwort-Hash eines Benutzers wiederhergestellt werden können.

5.5.1.5.2 NTLMv2

NTLM2 ist ein neueres NTLMSSP Signier- und Verschlüsselungsschema, das verwendet wird, wenn das Flag «Negotiate NTLM2 Key» gesetzt wurde. Die Ableitung dieser Verschlüsselungsschlüssel besteht aus vier Schritten:

1. Master Key Negotiation
2. Key Exchange
3. Key Weakening
4. Subkey Generation

Master Key Negotiation
Beim NTLM2-Signieren und -Verschlüsseln wird der User Session Key immer als Basis-Master Key verwendet. Wenn die NTLMv2-Authentifizierung verwendet wird, wird entweder der LMv2- oder der NTLMv2-User-Session Key als Master Key eingesetzt. Wenn die NTLMv1-Authentifizierung mit NTLM2-Sitzungssicherheit verwendet wird, wird der NTLM2 Session Response User Session Key als Master Key verwendet. Auch bei NTLMv2 wird das Passwort des Clients zur Berechnung (8.2.2 Keys) des Keys verwendet.

Die in NTLM2 verwendeten User Session Keys sind wesentlich stärker als jene in NTLM1 oder der LAN Manager Session Key, da sie sowohl die Challenge-Response als auch die Client-Nonce enthalten.

Key Exchange
Der Schlüsselaustausch für NTLM2 erfolgt wie zuvor für NTLM1 beschrieben. Auch hier wählt der Client einen 128-Bit-Random Key, verschlüsselt ihn mit dem Master Key mittels RC4 und sendet diesen im Feld «Session Key» der NTLM-Typ-3-Nachricht an den Server.

Key Weakening
Die Schlüsselschwächung in NTLM2 erfolgt einfach durch Abschneiden des erhaltenen Random Key auf die gewünschte Länge.

Der Random Key wird unter NTLM2 nur bei der Erzeugung der Subkeys für die Verschlüsselung geschwächt. Bei der Erzeugung der Signierschlüssel wird immer der volle 128-Bit Verschlüsselungsschlüssel verwendet.

Subkey Generation
In NTLM2 werden bis zu vier Subkeys von dem Random Key abgeleitet. Der Random Key wird nie zum Signieren oder Verschlüsseln von Nachrichten verwendet. Die Subkeys werden wie folgt abgeleitet:

1. Client-Signierschlüssel
   Der ungeschwächte 128-Bit Random Key wird mit der nullterminierten konstanten ASCII-Zeichenkette «session key to client-to-server signing key magic constant» verkettet. Darauf wird der MD5 Message-Digest-Algorithmus angewendet, was einen 16-Byte-Wert ergibt.
2. Server-Siginierschlüssel
   Der ungeschwächte 128-Bit Random Key wird mit der nullterminierten konstanten ASCII-Zeichenkette «session key to server-to-client signing key magic constant verkettet. Darauf wird der MD5 Message-Digest-Algorithmus angewendet, was einen 16-Byte-Wert ergibt.
3. Client-Verschlüsselungsschlüssel
   Der abgeschwächte (je nachdem, ob 40-Bit-, 56-Bit- oder 128-Bit-Verschlüsselung ausgehandelt wurde) Random Key wird mit der nullterminierten konstanten ASCII-Zeichenkette «session key to client-to-server sealing key magic constant» verkettet. Darauf wird der MD5 Message-Digest-Algorithmus angewendet, was einen 16-Byte-Wert ergibt.

4. Server-Verschlüsselungsschlüssel
   Der abgeschwächte (je nachdem, ob 40-Bit-, 56-Bit- oder 128-Bit-Verschlüsselung ausgehandelt wurde) Random Key wird mit der nullterminierten konstanten ASCII-Zeichenkette «session key to server-to-client sealing key magic constant» verkettet. Darauf wird der MD5 Message-Digest-Algorithmus angewendet, was einen 16-Byte-Wert ergibt.

5.5.2 Fazit

- Durch die Analyse des NTLM-Authentifizierungsprotokolls lässt sich feststellen, dass der von SPNEGO verwendete Verschlüsselungsschlüssel ein vom Client generierter Zufallsschlüssel ist. In NTLMv1 und NTLMv2 wird dieser Zufallsschlüssel durch einen «Key Exchange Key» verschlüsselt, der mit dem Passwort des Clients berechnet und an den Server übertragen wird.

- Für die Machbarkeitsanalyse des MitM bedeutet dies, dass der MitM das Passwort des Clients kennen muss, bevor der Server die NTLM CHALLENGE_MESSAGE an den Client sendet. Nur dann hat der MitM die Möglichkeit, den Key Exchange Key zu berechnen, der sowohl vom Client als auch vom Server berechnet wird. Nur in diesem Fall wäre es dem MitM möglich, die vom Client an den Server gesendete AUTHENTICATE_MESSAGE abzufangen und das Feld «Session Key» auszulesen. Er kann dann den ausgelesenen Wert (verschlüsselter Zufallsschlüssel) durch Kenntnis des Key-Exchange-Keys entschlüsseln und für die weiteren Schritte von SPNEGO verwenden, insbesondere für die Verschlüsselung des Public Key des Servers.

5.6 **Kerberos**

In diesem Kapitel wird das Authentifizierungsprotokoll Kerberos genauer analysiert. Kerberos ist eine von zwei Möglichkeiten, neben NTLM, sich beim RDP Verbindungsaufbau mit CredSSP zu authentifizieren.
Der Inhalt und die Ausführungen des folgenden Abschnitts beruhen sich auf folgenden Quellen: (Kryptowissen.de, 2016)

5.6.1 Funktionsweise

Bei Kerberos sind drei Parteien beteiligt: Client, Server (Ressource im Netzwerk, auf die der Client zugreifen will) und Kerberos-Server, auch Key Distribution Center genannt (Abbildung 16). Dieses Key Distribution Center besteht aus dem Authentication Server und dem Ticket-Granting-Server (TGS).
Wenn ein Benutzer auf eine Ressource zugreifen möchte, meldet er sich zuerst mit seinem Benutzernamen und Passwort beim Client an. Der Client gibt danach das Passwort in eine Hashfunktion und wandelt so das Passwort in einen geheimen Schlüssel um. Dann sendet der Client eine Anfrage an den Authentication Server für ein Ticket-Granting-Ticket. Diese Anfrage enthält unter anderem den Benutzernamen, den Namen des Dienstes, auf den der Client zugreifen möchte und eine Nonce.
Der Authentication Server sendet dann eine Antwort, die unter anderem das Ticket-Granting-Ticket enthält. Dieses Ticket-Granting-Ticket ist mit dem Master Key des Key Distribution Centers verschlüsselt. Das Ticket-Granting-Ticket enthält unter anderem den Benutzernamen,

die Adresse des Clients, den Session Key und die Gültigkeitsdauer. Der Client kann dieses Ticket-Granting-Ticket nicht einsehen, da es mit dem Master Key vom KDC verschlüsselt ist. Daneben enthält die Antwort auch noch einen Session-Key (Client – TGS), die Nonce aus der Anfrage, den Namen und einen Zeitstempel. Diese Informationen sind mit dem geheimen Schlüssel vom Client verschlüsselt, der vom Passwort abgeleitet wurde.
Danach sendet der Client eine Nachricht an den Ticket-Granting-Server. Diese Nachricht beinhaltet das Ticket-Granting-Ticket, den Namen der Ressource, auf welche der Client zugreifen will und einen Authenticator (Client-ID und Zeitstempel zusammengesetzt und mit dem Session-Key (Client – TGS) den der Client vorhin bekommen hat, verschlüsselt).
Der Ticket-Granting-Server antwortet mit einer Nachricht, welche das Ticket für die Ressource, den Client Namen, die Netzwerkadresse des Clients, ein Session-Key für die Kommunikation mit dem Server auf welchem sich die Ressource befindet (Client – Server), die Lebensdauer und einen Zeitstempel beinhaltet. Dieses Ticket ist mit dem Master Key der Ressource verschlüsselt. Daneben werden noch weitere Informationen mit der Nachricht übermittelt, wie der Session-Key zur Kommunikation zwischen Client und Ressource, Name der Ressource, Lebensdauer des Tickets und einen Zeitstempel. Diese Informationen sind mit dem Session-Key (Client – TGS) verschlüsselt. Dann kann der Client dem Server das Ticket sowie einen Authenticator, der mit

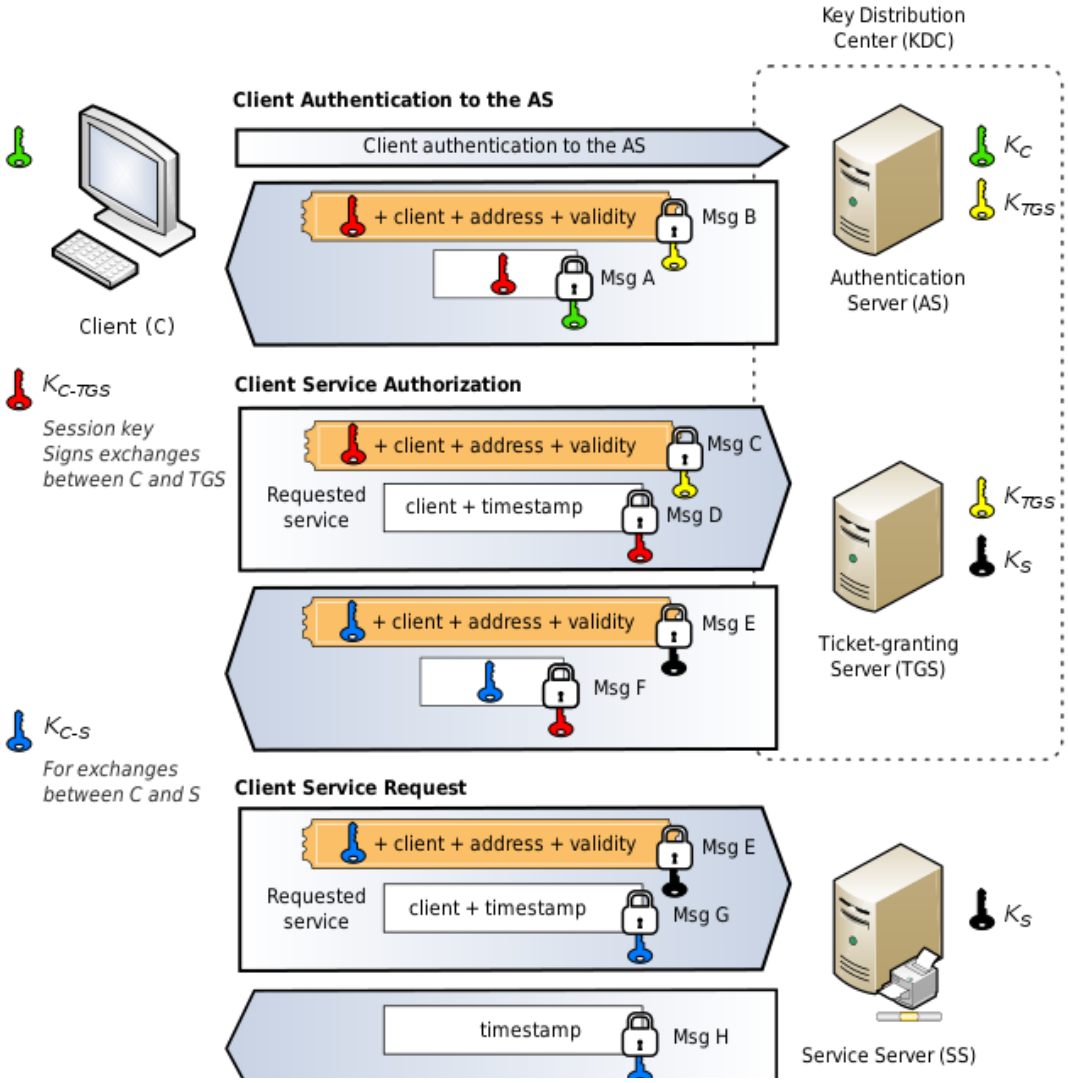


Abbildung 16: Kerberos Authentication (Wikipedia, 2021)

dem Session-Key für die Client-Server Kommunikation (Client – Server) verschlüsselt ist, senden. Danach bekommt der Client den Zugriff auf die Ressource.

5.7 **Evaluation**

Bei dieser Evaluation wird das CredSSP-Protokoll genauer mit dem NTLMv1 Authentifizierungsmechanismus betrachtet, da es im Vergleich zu NTLMv2 und Kerberos weniger sicherer ist. Die Machbarkeitsanalyse wurde zusätzlich mit Hilfe von Wireshark (5.7.1 Paketanalyse mit Wireshark) genauer analysiert. Am Ende dieses Kapitel wird bewertet, ob das Man-in-the-Middle über NLA/CredSSP Verbindung möglich ist oder nicht.

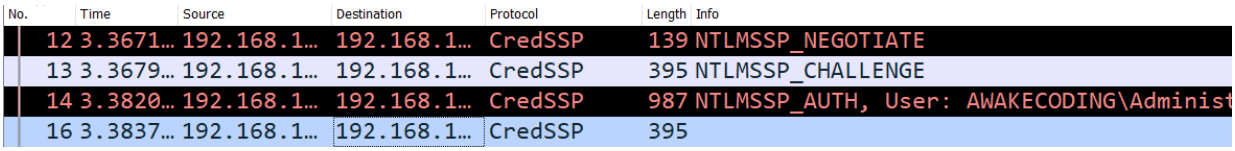
5.7.1 Paketanalyse mit Wireshark

Um eine genauere Analyse des RDP Verbindungsaufbaus und des CredSSP Authentifizierungsverfahrens durchzuführen, wurde zusätzlich eine Paketanalyse mithilfe von Wireshark durchgeführt. Diese soll helfen, um die bisherige theoretische Machbarkeitsanalyse zu verifizieren. Im Folgenden ist ein Wireshark-Capture (Abbildung 17) eines RDP-Verbindungsaufbaus zu sehen.

No.	Time	Source	Destination	Protocol	Length	Info
43	5.7279…	172.16.60.123	sinv-56045.edu.hsr.ch	TLSv1.2	237	Client Hello
44	5.7373…	sinv-56045.edu.hsr.ch	172.16.60.123	TCP	14…	ms-wbt-server(3389) → 5
45	5.7375…	sinv-56045.edu.hsr.ch	172.16.60.123	TLSv1.2	242	Server Hello, Certifica
46	5.7375…	172.16.60.123	sinv-56045.edu.hsr.ch	TCP	54	57082 → ms-wbt-server(3
47	5.7385…	172.16.60.123	sinv-56045.edu.hsr.ch	TLSv1.2	668	Client Key Exchange, Ch
48	5.7585…	sinv-56045.edu.hsr.ch	172.16.60.123	TLSv1.2	145	Change Cipher Spec, Fir
49	5.7595…	172.16.60.123	sinv-56045.edu.hsr.ch	TPKT	171	Continuation
50	5.7679…	sinv-56045.edu.hsr.ch	172.16.60.123	TPKT	331	Continuation
51	5.7694…	172.16.60.123	sinv-56045.edu.hsr.ch	TPKT	699	Continuation
52	5.7782…	sinv-56045.edu.hsr.ch	172.16.60.123	TPKT	171	Continuation
53	5.7789…	172.16.60.123	sinv-56045.edu.hsr.ch	TPKT	219	Continuation
54	5.7871…	sinv-56045.edu.hsr.ch	172.16.60.123	TPKT	123	Continuation
55	5.7874…	172.16.60.123	sinv-56045.edu.hsr.ch	RDP	571	ClientData
56	5.8060…	sinv-56045.edu.hsr.ch	172.16.60.123	TCP	54	ms-wbt-server(3389) → 5
57	5.8185…	sinv-56045.edu.hsr.ch	172.16.60.123	RDP	235	ServerData Encryption:
58	5.8187…	172.16.60.123	sinv-56045.edu.hsr.ch	T.125	123	erectDomainRequest
59	5.8188…	172.16.60.123	sinv-56045.edu.hsr.ch	T.125	123	attachUserRequest
60	5.8264…	sinv-56045.edu.hsr.ch	172.16.60.123	TCP	54	ms-wbt-server(3389) → 5

Abbildung 17: Wireshark Paketanalyse (Eigene Darstellung)

Wie hier ersichtlich, findet der TLS-Handshake zwischen den Paketnummern 43 - 48 statt und dann erfolgt die Authentifizierung im TPKT-Protokoll, die zwischen den Paketnummern 49 - 54 liegt. Nach einer erfolgreichen Authentifizierung wird das RDP-Protokoll und die verschiedenen Kanäle mit dem T.125-Protokoll aufgebaut. Da wir das TPKT-Protokoll nicht entschlüsseln konnten, wurde der CredSSP-Teil (Abbildung 18) in einem separaten Wireshark-Capture analysiert, welches wir im Internet gefunden haben.



No.	Time	Source	Destination	Protocol	Length	Info
12	3.3671…	192.168.1…	192.168.1…	CredSSP	139	NTLMSSP_NEGOTIATE
13	3.3679…	192.168.1…	192.168.1…	CredSSP	395	NTLMSSP_CHALLENGE
14	3.3820…	192.168.1…	192.168.1…	CredSSP	987	NTLMSSP_AUTH, User: AWAKECODING\Administ
16	3.3837…	192.168.1…	192.168.1…	CredSSP	395	

Abbildung 18: Wireshark CredSSP Capture (Quelle: (FreeRDP, 2021))

Nach dem TLS Handshake wird der CredSSP Teil durchgeführt. In diesem Wireshark Capture wurde der NTLM Authentifizierungsmechanismus genauer analysiert. Im folgenden Kapitel werden die analysierten Pakete in Sequenzdiagrammen genauer beschrieben und die wichtigsten Felder genauer erläutert.

5.7.2 Sequenzdiagramm

Das CredSSP-Protokoll mit NTLMv1 Authentifizierung wird anhand von zwei Sequenzdiagrammen genauer analysiert. Es wird ein Sequenzdiagramm ohne Man-in-the-Middle und eines mit Man-in-the-Middle beschrieben.

► Auf Grund der Lesbarkeit wird das Sequenzdiagramm «CredSSP mit NTLMv1 Authentifizierung mit MitM» im Anhang E: Sequenzdiagramm mit MITM nochmals grösser dargestellt.

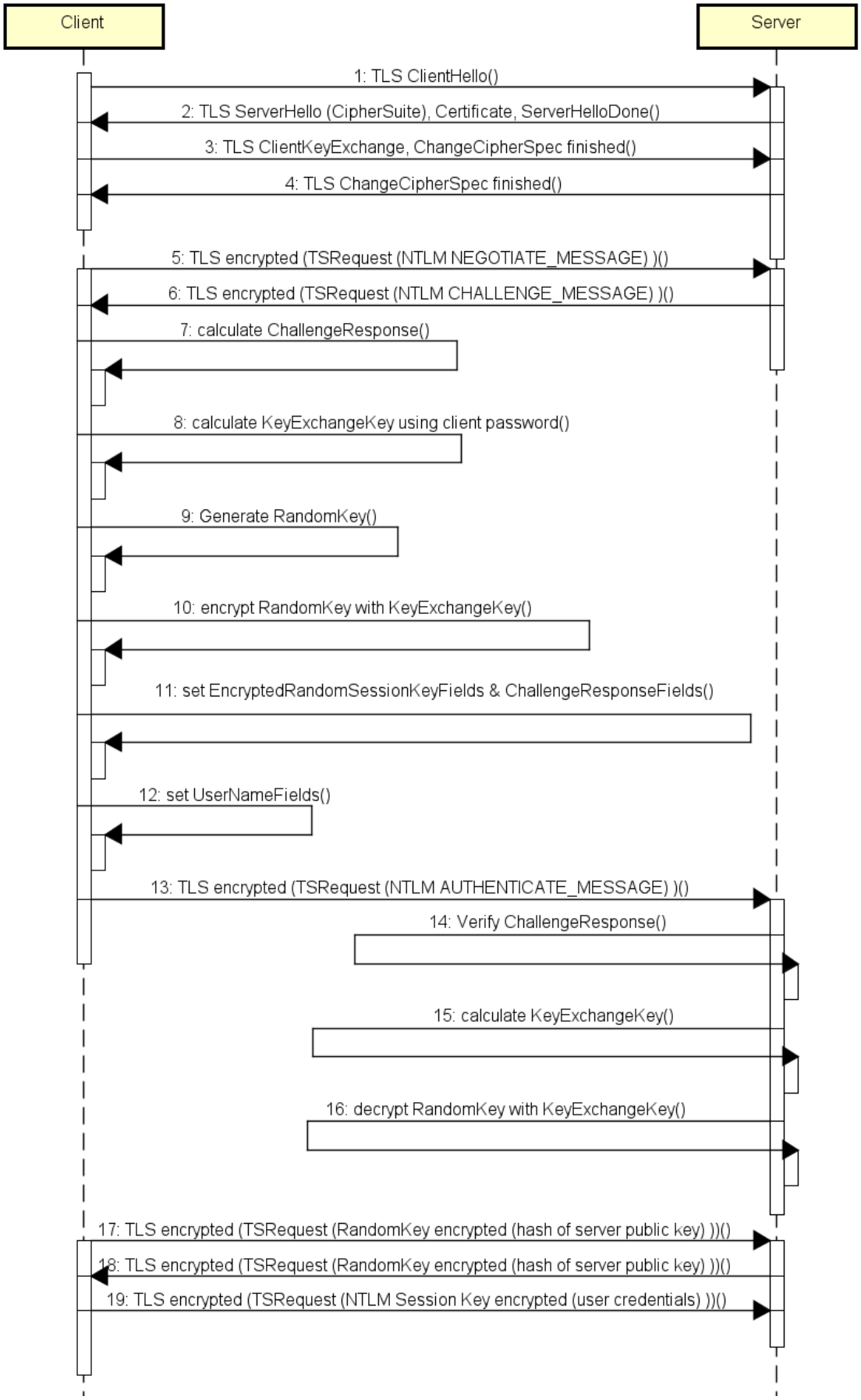


Abbildung 19: CredSSP mit NTLMv1 Authentifizierung (Eigene Darstellung)

5.7.2.1 CredSSP mit NTLMv1 Authentifizierung

In diesem Abschnitt wird das Sequenzdiagramm (Abbildung 19) ohne Man-in-the-Middle detailliert beschrieben.

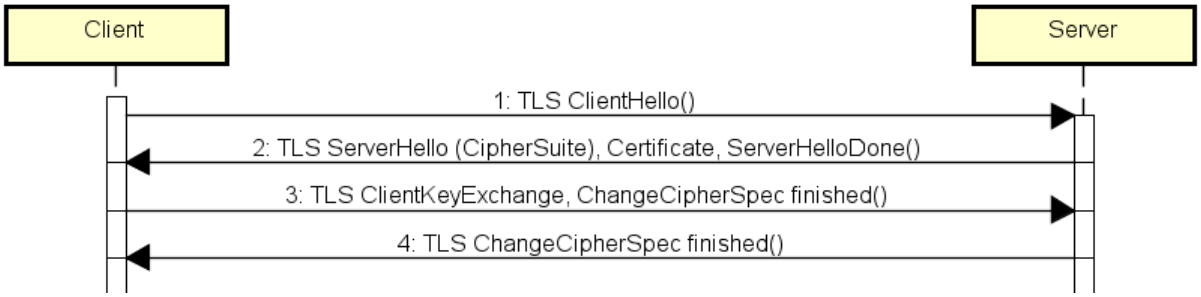


Abbildung 20: CredSSP Teil 1 (Eigene Darstellung)

Wie im CredSSP-Protokoll (5.3 CredSSP) beschrieben, wird während den Schritten **1-4** ein TLS-Handshake (5.2.2 Handshake) zwischen Client und Server durchgeführt. Nach dem TLS-Handshake werden alle CredSSP-Nachrichten über diesen verschlüsselten Kanal gesendet. (Abbildung 20)

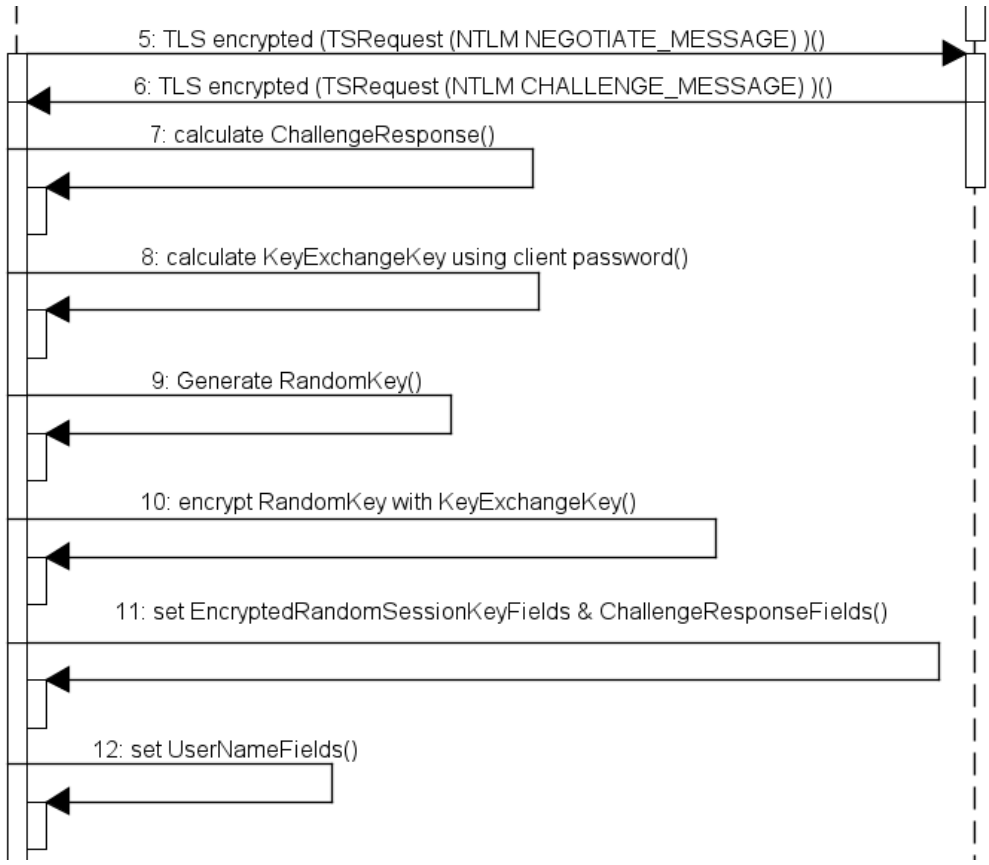


Abbildung 21: CredSSP Teil 2 (Eigene Darstellung)

Danach wird SPNEGO für die Benutzer und Serverauthentifizierung verwendet und wählt den entsprechenden Authentifizierungsmechanismus (Abbildung 21). In diesem Beispiel wurde das NTLMv1 Authentifizierungsmechanismus ausgewählt und besteht aus den folgenden drei Messages (5.5.1 Messages):

- NTLM-Negotiate Message
- NTLM-Challenge Message
- NTLM-Authenticate Message

Im **Schritt 5** wird die NTLM Authentifizierung mit der NTLM-Negotiate Message vom Client zum Server initiiert und gibt die gewünschten Sicherheitsmerkmale der Sitzung an.

Der Server sendet im **Schritt 6** eine NTLM-Challenge Message an den Client mit den vereinbarten Sicherheitsmerkmalen und der vom Server generierten Nonce.

In den **Schritten 7-12** werden nur die wichtigsten Felder und Berechnungen der NTLM-Authenticate-Message für die spätere Man-in-the-Middle-Analyse näher betrachtet. Alle anderen Felder können von Man-in-the-Middle an Client und Server weitergeleitet werden, ohne das sie manipuliert werden müssen.

Schritt 7 & 8 => Die beiden Berechnungen von ChallengeResponse und KeyExchangeKey werden clientseitig mit Hilfe des Client-Passwortes berechnet. Die ChallengeResponse Berechnungen werden zwischen LMChallengeResponse und NTLMChallengeResponse unterschieden.

Schritt 9 & 10 => In diesen Schritten wird clientseitig ein RandomKey (5.5.1.5 Ableitung des Verschlüsselungsschlüssels) erstellt, welcher dann mit dem vorher berechneten KeyExchangeKey verschlüsselt wird.

Schritt 11 & 12 => Die Felder «EncryptedRandomSessionKeyFields», «ChallengeResponseFields» und «UserNameFields» von der NTLM-Authenticate Message werden mit den vorherberechneten und generierten Werten gefüllt.

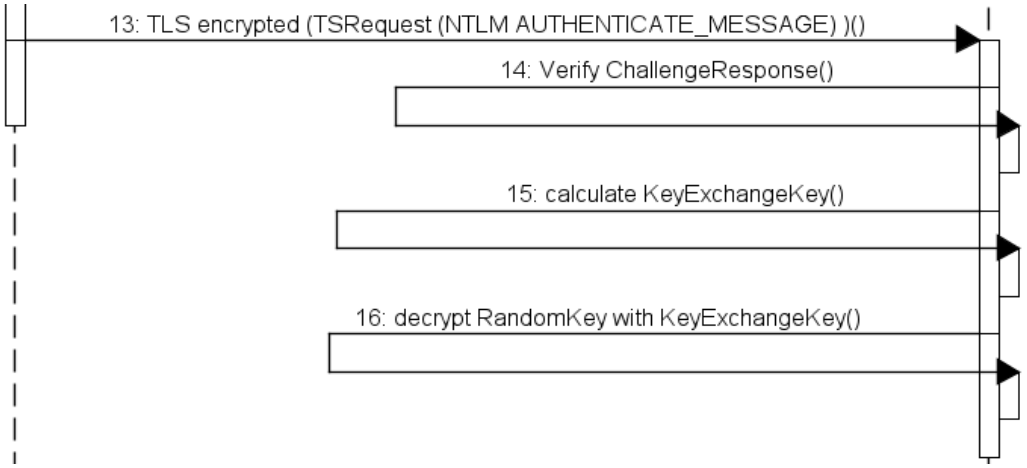


Abbildung 22: CredSSP Teil 3 (Eigene Darstellung)

In **Schritt 13** wird dann die NTLM-Authenticate Message vom Client an den Server gesendet, und in den **Schritten 14-16** kann der Server mit den gleichen Berechnungen serverseitig verifizieren, ob der Client das Passwort kennt, da der Server auch den Hash des Client-Passwortes kennt. (Abbildung 22)

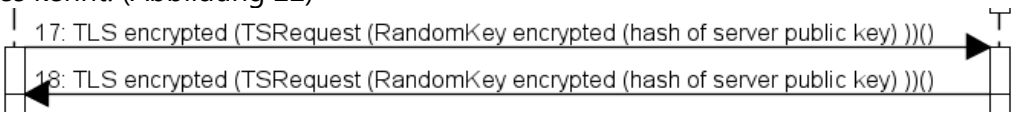


Abbildung 23: CredSSP Teil 4 (Eigene Darstellung)

In den **Schritten 17-18** wird der öffentliche Schlüssel vom vorherigen TLS-Handshake zwischen den beiden Parteien verifiziert, in dem öffentliche Schlüssel des Servers gehashed und mit dem Randomkey verschlüsselt über den TLS Kanal gesendet wird. Somit soll sichergestellt werden, dass kein Man-in-the-Middle Angreifer zwischen Client und Server sitzt. (Abbildung 23)

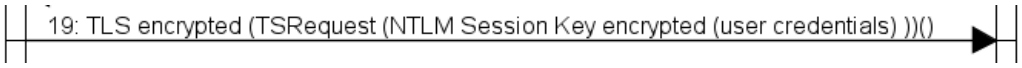


Abbildung 24: CredSSP Teil 5 (Eigene Darstellung)

Im letzten Schritt werden die Benutzeranmeldeinformationen mit dem RandomKey verschlüsselt vom Client an den Server gesendet. (Abbildung 24)

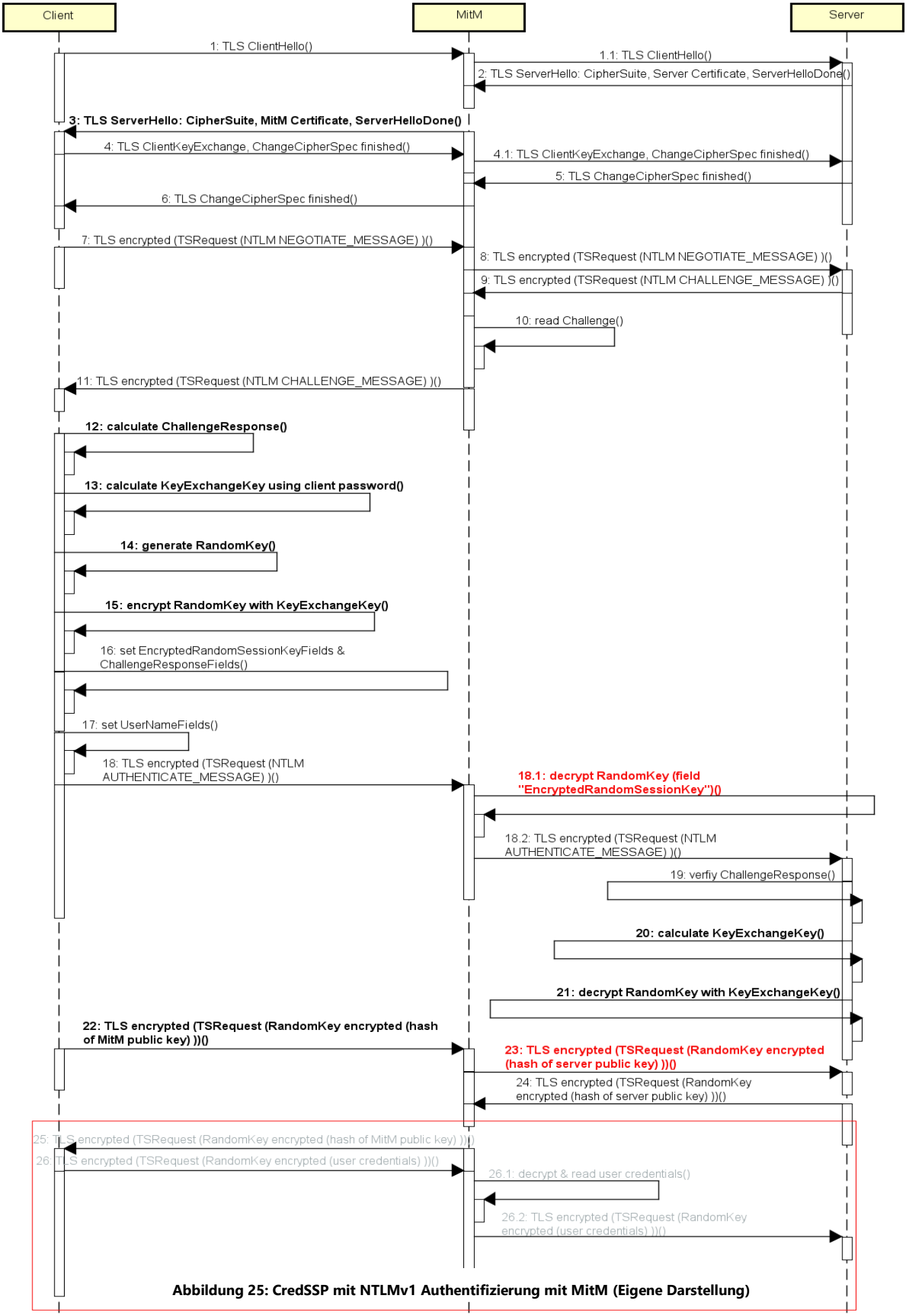


Abbildung 25: CredSSP mit NTLMv1 Authentifizierung mit MitM (Eigene Darstellung)

5.7.2.1 CredSSP mit NTLMv1 Authentifizierung mit MitM

In diesem Abschnitt wird das Sequenzdiagramm (Abbildung 25) mit Man-in-the-Middle näher betrachtet und beschrieben, was der Man-in-the-Middle für den Angriff manipulieren muss.

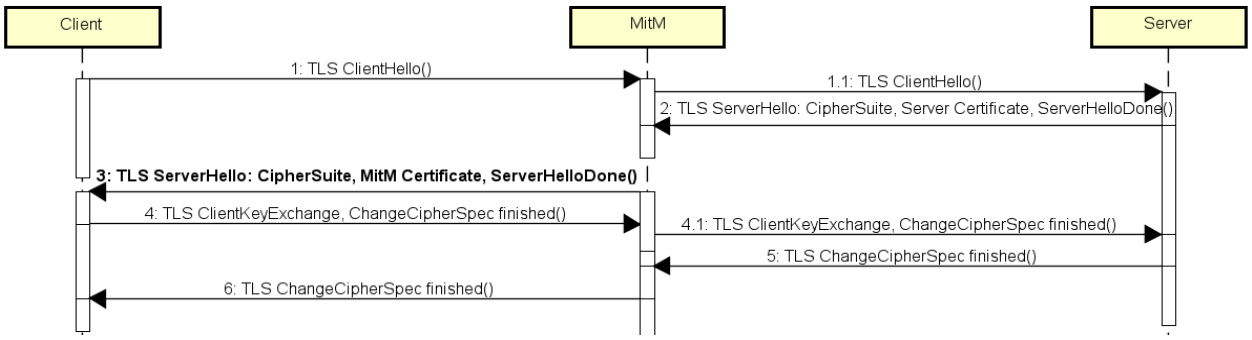


Abbildung 26: CredSSP mit MitM Teil 1 (Eigene Darstellung)

Beim TLS-Handshake muss der Man-in-the-Middle das Server-Zertifikat mit seinem eigenen Zertifikat austauschen und an Client weiterleiten, damit er den Traffic zwischen Client und MitM lesen kann (siehe Schritt 3 Abbildung 26)

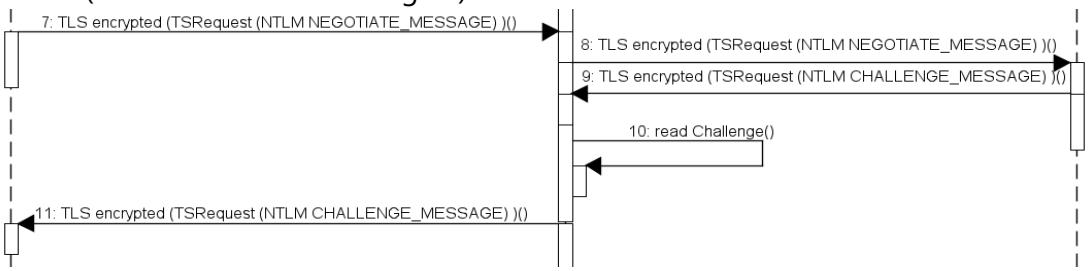


Abbildung 27: CredSSP mit MitM Teil 2 (Eigene Darstellung)

Beim SPNEGO Handshake können die ersten zwei Messages von NTLMv1 ohne Probleme weitergeleitet werden. (siehe Schritte 7 – 11 Abbildung 27)

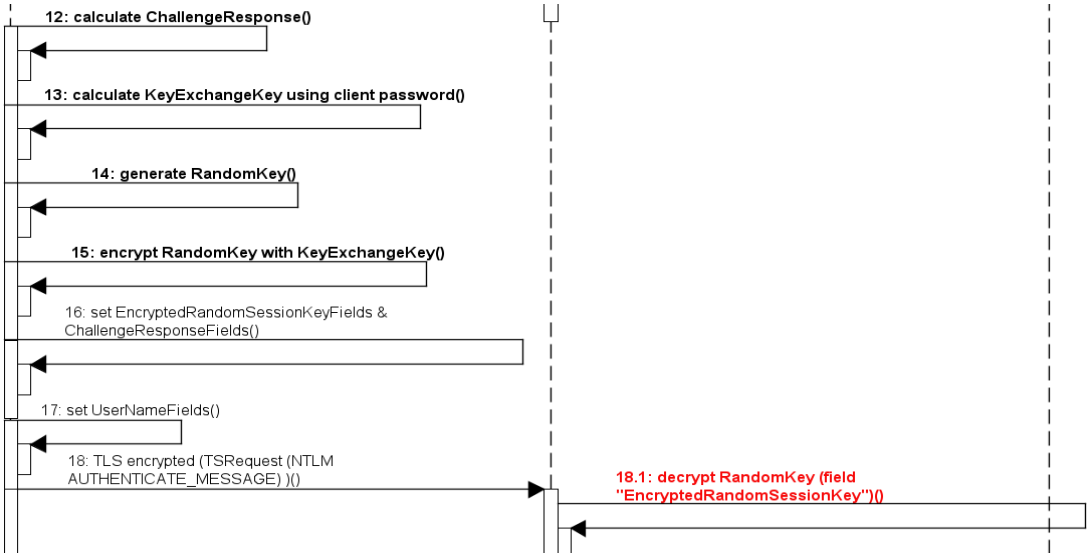


Abbildung 28: CredSSP mit MitM Teil 3 (Eigene Darstellung)

In Schritt 18 schlägt der Man-in-the-Middle fehl oder kennt den RandomKey nicht, weil er mit KeyExchangeKey verschlüsselt ist und kann somit den Traffic nicht entschlüsseln. Der Man-in-the-Middle kennt das Client-Passwort nicht, weil es bei diesem Authentifizierungsmechanismus nicht über den Kanal gesendet wird. (Abbildung 28)

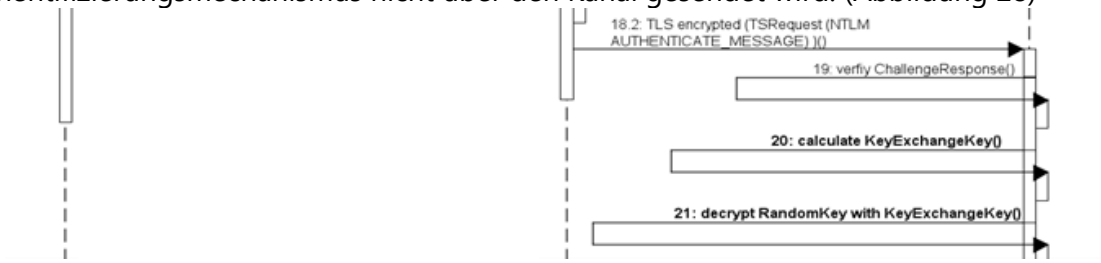


Abbildung 29: CredSSP mit MitM Teil 4 (Eigene Darstellung)

Wenn die NTLM-Authenticate-Message ohne Manipulation des MitM weitergeleitet wird, kann der Server erfolgreich verifizieren, andernfalls schlägt die Verifizierung fehl und der Server gibt eine Fehlermeldung zurück. (Abbildung 29)

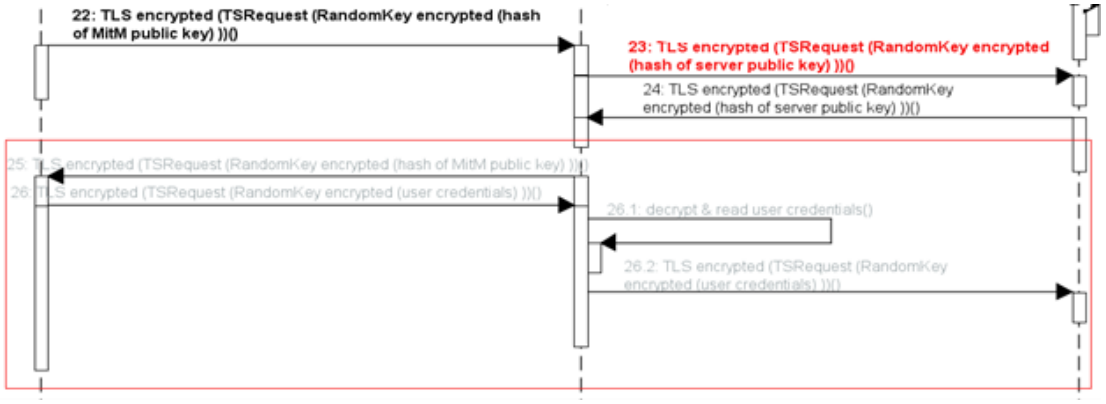


Abbildung 30: CredSSP mit MitM Teil 5 (Eigene Darstellung)

In Schritt 23 scheitert der Man-in-the-Middle erneut, weil er nicht an den Zufallsschlüssel herankommt, da dieser mit dem KeyExchangeKey verschlüsselt ist, welcher wiederum vom Client Passwort abgeleitet wurde. Die grau markierten Stellen werden nicht mehr ausgeführt, weil der Public Key des Server-Zertifikates nicht übereinstimmt, mit dem Public Key der für die TLS-Verbindung benutzt wurde, da der Man-in-the-Middle sein eigenes Zertifikat hat und somit einen anderen Public Key, und der Server die Verbindung trennt. (Abbildung 30)

5.7.3 Fazit

► Nach dieser Analyse wurde festgestellt, dass ein Man-in-the-Middle über eine direkte NLA/CredSSP-Verbindung nicht möglich ist. Der Grund ist das Client-Passwort (siehe Schritt 18), das dem Man-in-the-Middle nicht bekannt ist. Bei Authentifizierungsmechanismen wie Kerberos oder NTLM ist das Kennwort nur dem Server und dem Client bekannt und wird nicht über den Kanal gesendet. Es wird zur Verifizierung und Berechnung von Client und Server verwendet und ist notwendig für die Man-in-the-Middle Attacke. Bei der Analyse wurde jedoch ein weiterer möglicher funktionierender Man-in-the-Middle-Angriff identifiziert, den wir versuchen werden in Python zu implementieren. Dieser Angriff wird im nächsten Kapitel «Alternative MitM-Variante» ausführlicher beschrieben.

5.8 **Aufgetretene Probleme**

5.8.1 Wiresharkanalyse

In diesem Kapitel werden die Probleme bei der Inspektion (9.4 Wireshark - CredSSP Paketanalyse) des RDP-Datenverkehrs mit Wireshark beschrieben.

- **Der private Schlüssel** konnte nicht aus dem vorhandenen Zertifikat extrahiert werden, da das Passwort von Zertifikat nicht bekannt war, und daher wurde ein selbstsigniertes Zertifikat mit dem Tool openSSL erstellt.
- **Server musste neu konfiguriert werden**, damit das oben selbst erzeugte Zertifikat als Standardzertifikat für RDP-Verbindungen benutzt wird. Damit konnte später der TLS-Kanal entschlüsselt und in Wireshark angezeigt werden.
- **Perfect Forward Secrecy** generiert mehrere Sitzungsschlüssel und das Ziel ist, dass der Datenverkehr nicht nachträglich entschlüsselt werden kann, selbst wenn der private Schlüssel bekannt ist. Daher mussten die vorhandene Cipher-Suite-Liste auf dem Windows Server angepasst werden, damit Perfect Forward Secrecy nicht mehr unterstützt wird und danach der RDP-Verkehr mit dem privaten Schlüssel aus dem Server-Zertifikat in Wireshark entschlüsseln werden kann.

5.9 **Alternative Man-in-the-Middle Variante**

Da wir während der Analyse zum Schluss gekommen sind, dass man eine NLA/CredSSP Verbindung nicht intercepten kann, da die Session Keys vom Passwort des Clients abgeleitet werden und dieses nicht über das Netz übertragen wird, und danach der Public Key des Servers mit dem Session Key verschlüsselt und verifiziert wird, haben wir nach einer anderen Möglichkeit gesucht, um eine RDP Verbindung, die mit NLA/CredSSP geschützt ist, trotzdem zu intercepten. Das eigentliche Problem ist, dass wir das Client Passwort nicht kennen und somit die Session-Keys nicht ableiten können. Deshalb haben wir uns überlegt, wie wir an dieses Passwort kommen können, und sind auf die Idee gekommen, dass wir dem Client einen Fake-Login Screen anzeigen können. Dieser Fake-Login Screen wird beim Client dann als erstes angezeigt und der Client kann dann dort das Passwort eingeben, und so kommen wir an das Passwort und dann können wir damit zum Server eine mit NLA/CredSSP geschützte Verbindung aufbauen (Abbildung 31).

5.9.1 Beschreibung

Da man, wie oben in der Analyse beschrieben, keine NLA/CredSSP-Verbindung vom Client über den MitM zum Server aufbauen kann, da dem MitM das Passwort nicht bekannt ist und die abgeleiteten und verschlüsselten Session-Keys benötigt werden, um im CredSSP Teil den Public Key des Servers zu verifizieren, bauen wir in dieser Variante statt vom Client über den MitM zum Server, nur vom MitM zum Server eine NLA/CredSSP Verbindung auf. So gibt es keine Probleme mehr mit dem Public Key des Servers, da der Key bei der TLS Verschlüsselung und bei der CredSSP Authentifizierung derselbe ist. Vom Client zum MitM wird stattdessen eine Verbindung mit Enhanced Security aber ohne CredSSP aufgebaut. So muss sich der Client erst nach Aufbau der Verbindung authentifizieren und da dann die RDP Channels bereits aufgebaut wurden, kann man das Passwort abfangen. Damit man an das Passwort vom Client kommt, wird dann nach Aufbau der Verbindung zwischen Client und MitM ein Fake-Login-Screen angezeigt, der aussieht wie ein Standard-Anmeldebildschirm von Windows, bei dem der Client dann den Benutzernamen und das Passwort eingeben kann und dieses wird dann an den MitM übermittelt resp. die Tastenanschläge werden aufgezeichnet und daraus wird das Passwort konstruiert.

Wenn der MitM dann das Passwort hat, kann er eine CredSSP Verbindung zum Server aufbauen. Dieses Mal ist das kein Problem, da man das Passwort kennt und so die Session Keys berechnen kann und das PubAuthKey Feld in CredSSP richtig füllen kann, so dass der Server die Verbindung nicht beendet. Wenn diese Verbindung steht, leitet der MitM die eingehenden Pakete vom Server an den Client weiter und die eingehenden Pakete vom Client an den Server weiter. Der MitM muss dafür alle Pakete entschlüsseln mit den ausgehandelten Schlüsseln und muss es mit dem richtigen Schlüssel wieder verschlüsseln.

Diese Methode unterscheidet sich zur ursprünglichen Variante, da nur zwischen dem Server und dem MitM eine NLA/CredSSP Verbindung aufgebaut wird und nicht vom Client über den MitM zum Server. Da die Enhanced Security RDP Verbindung ohne NLA/CredSSP keine Mechanismen zur Sicherstellung, dass kein Man-in-the-Middle dazwischen ist, hat, kann man mithilfe eines angezeigten falschen Login Screen das Passwort abfangen und kann dieses dann verwenden um eine NLA/CredSSP Verbindung zum Server aufzubauen.

Mit dieser Variante kann man auch einen Man-in-the-Middle Angriff starten, wenn auf dem Server NLA aktiviert ist resp. erzwungen wird.

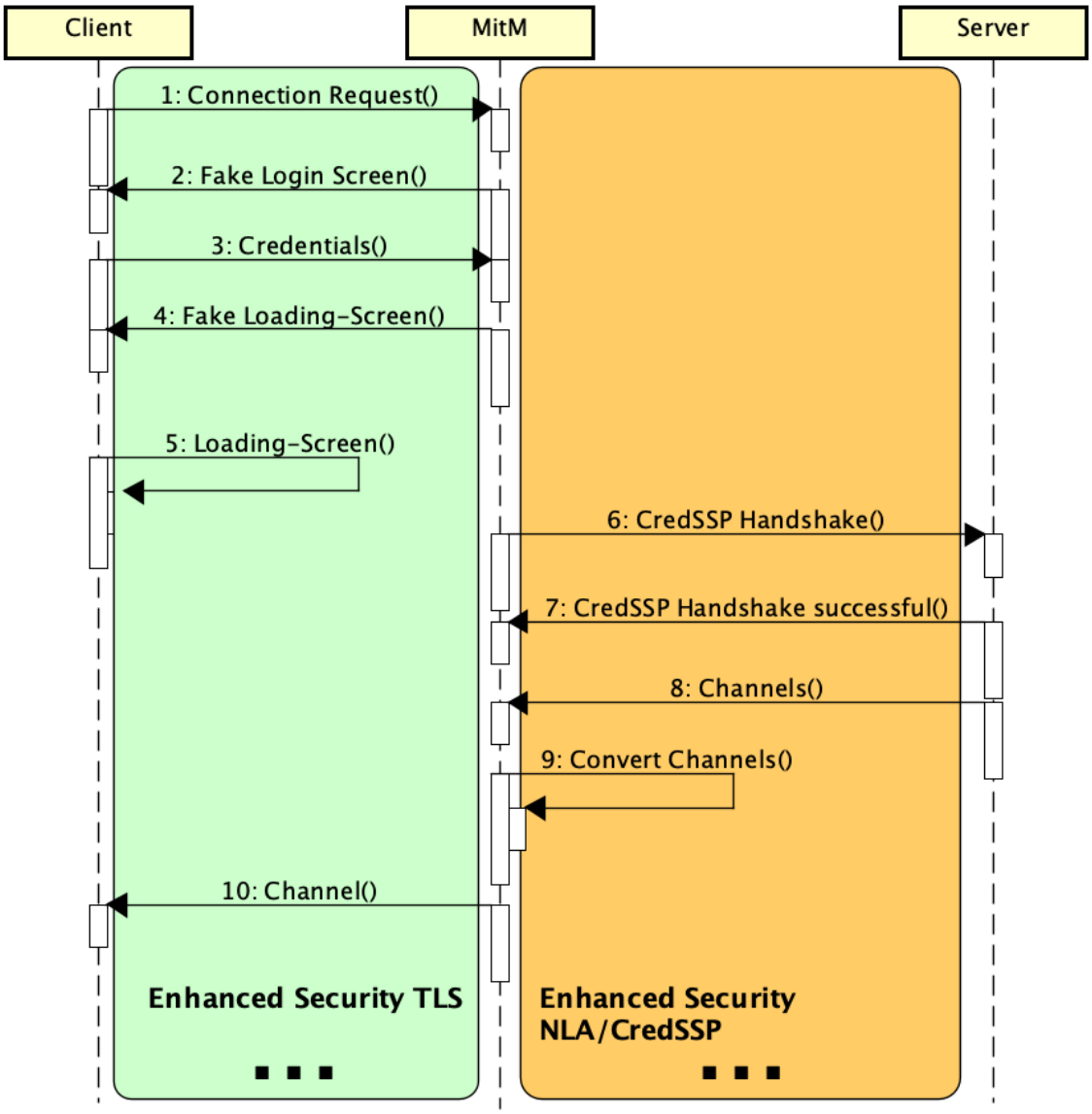


Abbildung 31: Alternative MitM-Variante (Eigene Darstellung)

6 Analyse: PyRDP

In diesem Kapitel wird der PyRDP-Code genauer analysiert. Es wird hier grob die Codestruktur von PyRDP, die verwendeten Libraries und die wichtigsten Methoden und Klassen beschrieben, welche für diese Arbeit relevant sind.

6.1 **Code-Struktur**

Die Code-Struktur von PyRDP ist in mehrere Module aufgeteilt. Diese werden im Folgenden grob beschrieben bezüglich ihres Zwecks und ihrer Abhängigkeiten.
(Aufgrund der Anzahl der vorhandenen Klassen und der komplexen Struktur kann das Tool nicht in Form eines Klassendiagramms dargestellt werden. Da die Abhängigkeiten der einzelnen Module untereinander stark ausgeprägt sind, ist es auch nicht möglich, ein lesbares Package-Diagramm aufzuzeigen.)

Der Inhalt und die Ausführungen des folgenden Abschnitts beruhen sich auf folgenden Quellen: (pyrdp, 2021)

Packgage	Beschreibung	Abhängigkeit
core	Es enthält Code, der im gesamten Projekt verwendet wird. Dabei handelt es sich um eigenständige Primitive und Hilfsfunktionen, die wiederverwendbar sind und die folgenden Aufgaben enthalten: • Kodierung und Dekodierung • Hilfsfunktionen für Stream-Parsing und Konvertierung von Bytes in String und umgekehrt • Verwaltung der Konfigurationsdateien • Twisted Primitive • Abstrakte Klassen und Schnittstellen sowie Basistypen	• enum • logging • mitm • pdu
enum	Es stellt Konstanten für das RDP-Protokoll bereit. Dazu gehören: • RDP Nachrichtentypen • Tastatur-Scan-Codes • Bitfeld-Flags für Nachrichten und Protokolldateneinheiten (PDUs).	
layer	Es ist der Kern des PyRDP Netzwerkstapels. Die verschiedenen Protokolle von RDP sind in verschiedene Layers unterteilt und wird auch für die Abstraktion auf der Festplatte verwendet.	• core • enum • pdu • parser • logging • security
logging	Es ist für die Protokollierung zuständig und soll für eine konsistente Protokollierung sorgen.	• enum • layer • parser

		• pdu • security
mcs	Es bietet eine Abstraktion für die verschiedenen Kanäle von RDP.	• layer • pdu
mitm	Es ist die Hauptkomponente und ist für das Abfangen und Manipulieren von Verbindungen zuständig. Für jede Schicht der Protokolle gibt es Abfangfunktionen, Hooks zum Überprüfen und Ändern von Daten. Außerdem stellt sie eine Anwendungslogik für den MITM-Server und den Verbindungsaufbau mit dem Zielserver bereit.	• enum • layer • parser • pdu • logging • core • recording • mcs • security
pdu	Es zentralisiert Klassen zur Darstellung der unterstützten RDP-Nachrichtentypen. Diese PDUs können von einer Verbindung gelesen und geschrieben werden und sind die Hauptansicht des RDP-Datenstroms.	• enum
parser	Es ist für die RDP-Implementierung zuständig. Es ist für das Wire-Protokoll zuständig, das von rohen Bytes in PDUs und von PDUs zurück in rohen Bytes umwandelt. Parser werden nach der Protokollschicht von RDP unterteilt, auf der sie arbeiten.	• core • enum • pdu • security • logging
player	Es enthält Code und Logik für das GUI von PyRDP und ist für die Darstellung der aktuellen RDP-Sitzung in verschiedenen Formen verantwortlich.	• parser • ui • logging • core • enum • pdu • mitm • layer
recording	Es bietet die Möglichkeit, eine RDP-Sitzung zur späteren Wiedergabe auf die Festplatte zu exportieren.	• enum • layer • pdu • logging • parser
security	Es regelt die im RDP-Protokoll verwendete Kryptografie wie z.B. RSA und RC4.	• enum • core • layer • logging • parser • pdu
ui	Es enthält zusätzliche Primitive, die zum Player-Modul gehören.	• logging

Tabelle 2: PyRDP – relevante Packages

Für diese Arbeit ist das mitm-Modul das wichtigste, da dort die Verbindung zum effektiven Server hergestellt wird und die verschiedenen Kanäle von RDP aufgebaut werden.

6.1.1 Observer Pattern

Das PyRDP-Projekt verwendet das Observer-Pattern, um über Zustandsänderungen zu informieren und die abhängigen Objekte automatisch zu aktualisieren. Da es sich um ein Man-in-the-Middle-Tool handelt, ist es wichtig, über die verschiedenen Änderungen von Client und Server informiert zu werden.

Das Pattern wird verwendet, um die verschiedenen Nachrichten aus den Layern von RDP wie z. B. X224, MCS zu erkennen. Die Nachrichtentypen der verschiedenen Layers von RDP werden in einem Enum gespeichert. Für jeden Layer gibt es einen Observer und dieser wird mit der Enum-Datei des entsprechenden Layers gemappt. Beim Empfang der PDU wird geprüft, ob eine der Nachrichten mit der in den Enum-Dateien definierten Layer übereinstimmt. Wenn sie übereinstimmen, fährt das PyRDP-Projekt mit dem entsprechenden Layer fort und sendet die Bytes an Client oder Server. Es wartet dann auf die passende PDU für weitere Aktionen.

6.2 **Verwendete Libraries**

Hier beschreiben wir kurz die wichtigsten Bibliotheken des PyRDP-Projektes, die bereits vorhanden waren, für die Erweiterung zusätzlich hinzugefügt wurden und bei der Analyse berücksichtigt, aber nicht verwendet wurden.

Library	Beschreibung
Twisted	Es ist ein asynchrones Netzwerkframework für Internetanwendungen für Client und Server. Es unterstützt auf unterschiedliche Weise TCP, UDP, SSL/TLS, IP-Multicast, Unix-Domain-Sockets, HTTP, SMTP und viele weitere Protokolle. (PyPi, 2021)
OpenSSL	Es ist ein Wrapper-Modul um die OpenSSL-Bibliothek und enthält Methoden und Objekte für die SSL-Verbindung. Zusätzlich enthält es die Fehlercodes von OpenSSL für den umfangreichen Fehlerbehandlungsmechanismus. (PyPi, 2021)
Impacket	Es ermöglicht Zugriff auf die Netzwerkpakete und kann einfache und konsistente Weise Netzwerkpakete erstellen und zuweisen. (PyPi, 2021)
TKINTER	Es ist ein GUI-Toolset und Widget Library für Python. (PyPi, 2021)
requests-credssp	Es ermöglicht eine CredSSP Authentifizierung auf HTTPS und unterstützt NTLM und Kerberos. (PyPi, 2021)

Tabelle 3: PyRDP - verwendete Libraries

Im folgenden Abschnitt wird die Verwendung im PyRDP Code beschrieben.

6.2.1 Twisted

Der Kern der Twisted Library ist der Reaktor für Event Loops. Er wartet auf Ereignisse oder Nachrichten von Client oder Server und leitet sie weiter. Es bietet Schnittstellen für Netzwerkkommunikation, Threading und Eventdispatcher. Hauptsächlich wird es für den TCP Verbindungsaufbau verwendet.

6.2.2 OpenSSL

Es wird für die TLS Verbindung und für das Kreieren von Private Key und Public Key verwendet.

6.2.3 Impacket

Diese Library wurde zusätzlich für die CredSSP-Authentifizierung hinzugefügt, die später in Kapitel 7.1 Fake Login Screen näher beschrieben wird. Es bietet Funktionen für die GSS-API, SPNEGO und NTLM.

6.2.4 TKINTER

Diese Library wurde für die Implementierung von dem Fake-Login-Screen verwendet, welche später im Kapitel 7.1 Fake Login Screen genauer beschrieben wird.

6.2.5 requests-credssp

Es unterstützt auch die CredSSP-Authentifizierung, da die dieses Library nur über die HTTPS verwendet wird, wurde diese bei der Analyse abgelehnt.

6.3 **Wichtigste Methoden/Klassen**

Hier werden die wichtigsten Klassen und Methoden kurz beschrieben, die später für die alternative Man-in-the-Middle-Variante angepasst werden müssen.
Für die alternative Man-in-the-Middle-Variante ist es wichtig, das Passwort über einen Fake-Login Screen vom Client auszulesen (siehe 5.9 Alternative Man-in-the-Middle Variante). Es wird in den folgenden Klassen und Methoden aus dem PyRDP-Projekt ausgelesen:

BasePathMITM.py (Methode=>handleClientLogin)

```
def loginAttempt(self):
    if self.state.loggedIn or self.state.inputBuffer == "":
        return
    self.state.credentialsCandidate = self.state.inputBuffer
    self.state.inputBuffer = ""
    self.log.info("Credentials attempt from heuristic: %(credentials_attempt)s", {
        "credentials_attempt": (self.state.credentialsCandidate)
    })
```

DeviceRedirectionMITM.py (Methode=>handleClientLogin)

```
    def handleClientLogin(self):
        """
        Handle events that should be triggered when a client logs in.
        """
        if self.state.credentialsCandidate or self.state.inputBuffer:
            self.log.info("Credentials candidate from heuristic: %(credentials_candidate)s", {
                "credentials_candidate" : (self.state.credentialsCandidate or self.state.inputBuffer)
            })
        # Deactivate the logger for this client
        self.state.loggedIn = True
        self.state.shiftPressed = False
        self.state.capsLockOn = False
```

```
        self.state.credentialsCandidate = ""

        self.state.inputBuffer = ""
```

Das bestehende PyRDP-Projekt verwendet das Enhanced Security TLS, um sich mit dem Server zu verbinden, aber diese TLS-Verbindung muss durch die CredSSP-Verbindung ersetzt werden, damit sie mit NLA funktioniert. In der folgenden Klasse und Methoden wird die TLS-Verbindung gestartet und die Verbindung zum Server aufgebaut:

RDPMITM.py (Methode=>startTLS)

```
 def startTLS(self, onTlsReady: typing.Callable[[], None]):
        """
        Execute a startTLS on both the client and server side.
        """
        self.onTlsReady = onTlsReady


        # Establish TLS tunnel with target server...
        contextForServer = ClientTLSContext()
        self.server.tcp.startTLS(contextForServer)


        # Establish TLS tunnel with client.
        reactor.callLater(1, self.doClientTls)
```

RDPMITM.py (Methode => connectToServer)

```
    async def connectToServer(self):
...
        reactor.connectTCP(self.config.targetHost, self.config.targetPort, serverFactory)
...
```

7 Implementierung der PyRDP-Erweiterung

7.1 **Fake Login Screen**

7.1.1 Implementation

Da wir in der Analyse zum Schluss gekommen sind, dass ein direkter Man-in-the-Middle Angriff auf eine mit NLA/CredSSP geschützte Verbindung nicht möglich ist, haben wir uns eine alternative MitM-Variante überlegt (siehe Kapitel 5.9 Alternative Man-in-the-Middle Variante). Für diese Variante benötigen wir das Passwort vom User, bevor wir eine CredSSP geschützte Verbindung vom Man-in-the-Middle zum Server aufbauen können. Damit wir das erreichen können, implementieren wir einen Fake-Login-Screen, welcher dem Client dann angezeigt wird und er da sein Passwort eingeben kann und wir so an das Passwort kommen.
Da das Tool «PyRDP» erweitert wird und dieses ist in Python geschrieben, wurde der Fake-Login-Screen ebenfalls in Python geschrieben. Um den Fake-Login-Screen zu implementieren, wurde das GUI-Toolkit Tkinter für Python verwendet.
Für den Hintergrund des Login Screens haben wir ein Standardbild von Windows 10 verwendet, welches sich auf jedem Windows 10 Home Betriebssystem unter C:\Windows\Web\Screen befindet. Das Bild wurde dann noch etwas abgedunkelt, damit es aussieht wie der originale Login Screen. Danach haben wir dem Bild den Standardavatar von Windows hinzugefügt. Dieser befindet sich unter C:\ProgramData\Microsoft\User Account Pictures. Bei diesem Bild musste der Hintergrund entfernt werden, damit er transparent ist und das Bild musste auch noch rund ausgeschnitten werden. Ebenfalls wurde unter dem Avatarbild, wo normalerweise der Username steht, der Text «other user» hinzugefügt. Dieses Bild wurde dann im Python Skript, welches den Fake-Login-Screen (Abbildung 32) darstellt, als Hintergrund festgelegt und so eingestellt, dass sich das Bild automatisch der Grösse des Fensters anpasst.

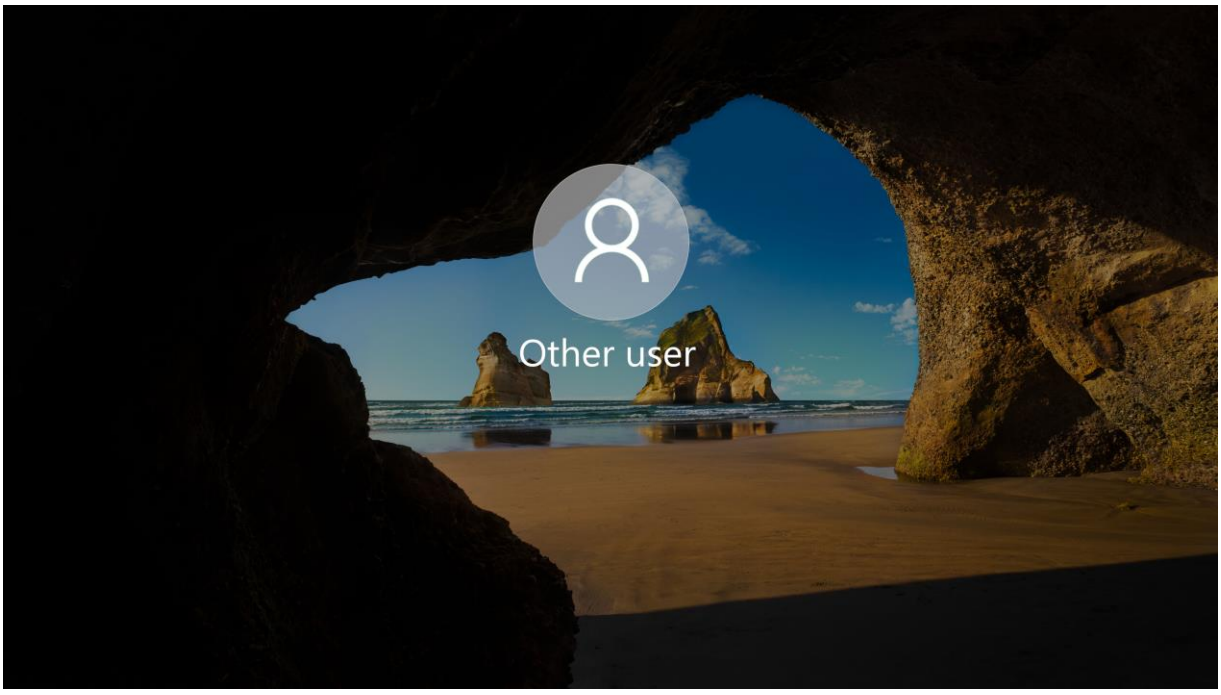


Abbildung 32: Hintergrundbild des Fake-Login-Screen (Eigene Darstellung)

Für die beiden Eingabefelder für Benutzername und Passwort wurden zwei Entry Elemente von Tkinter benutzt. Diese wurden unter dem «Other user» Text platziert. Als Schriftart wurde Segoe UI verwendet. Für den Login Button wurde ein Button Element benutzt und als Hintergrund für diesen Button wurde ein zugeschnittener Screenshot eines Buttons auf einem originalen Lock Screen verwendet. Dieser Login Button wurde am rechten Ende des Eingabefeldes für das Passwort platziert. (siehe Abbildung 33)

```
txt_username = Entry(window, font=("Segoe UI", 13), bd=2, bg="white", insertofftime=600, insertwidth
="1p", highlightthickness=1)

txt_username.config(highlightbackground = "gray", highlightcolor= "#eaeaea")



txt_username.place(relx=0.5, rely=0.61, anchor=CENTER, height=40, width=290)

txt_username.focus()

txt_password = Entry(window, show="•", font=("Segoe UI", 20), bd=2, bg="white", insertofftime=600, i
nsertwidth="1p", highlightthickness=1)

txt_password.config(highlightbackground = "gray", highlightcolor= "gray")

txt_password.place(relx=0.489, rely=0.675, anchor=CENTER, height=40, width=259)

txt_password.configure(cursor="xterm")

#Login Button

login_image = PhotoImage(file = "images/LoginButton.png")

btn_login = Button(window, text = "Login", image=login_image, command=clicked, width=33, height=
33, highlightthickness=1)

btn_login.config(highlightbackground = "gray", highlightcolor= "gray")

btn_login.place(relx=0.57, rely=0.653)
```

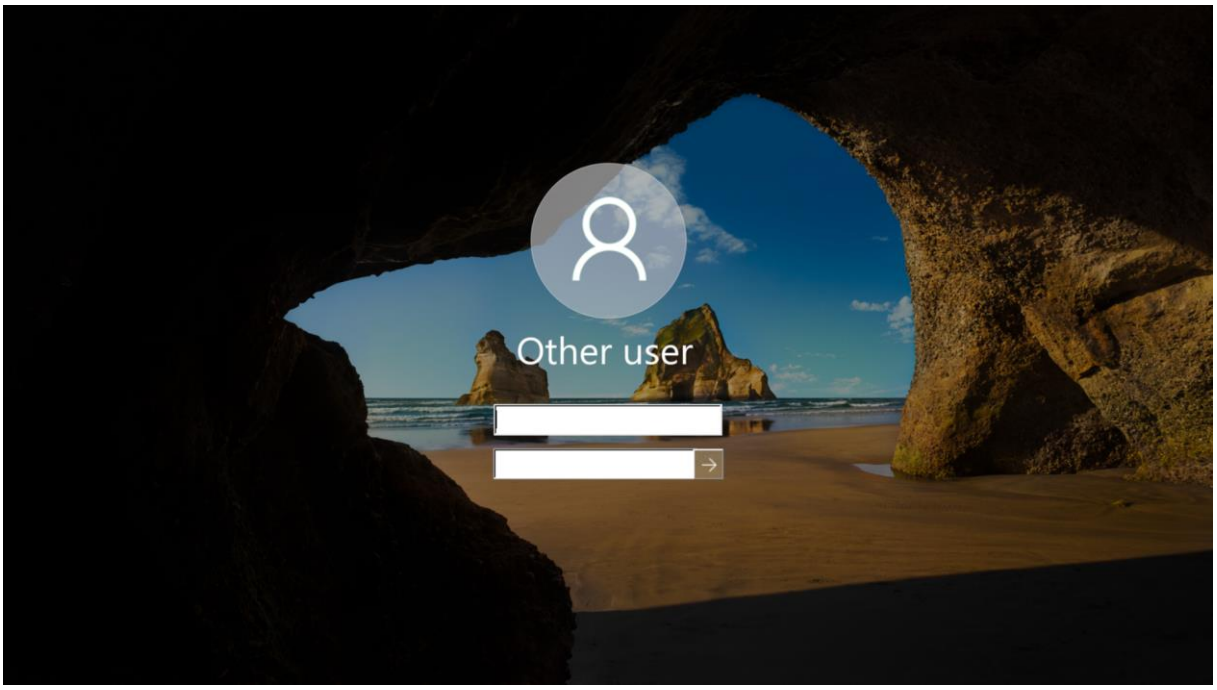


Abbildung 33: kompletter Fake-Login-Screen (Eigene Darstellung)

Damit die Verbindung zum Server über CredSSP hergestellt werden kann, nachdem sich der Client beim Fake-Login-Screen «angemeldet» hat und der Client nicht die ganze Zeit beim Fake-Login-Screen bleibt, haben wir noch einen Loading Screen eingefügt. Dieser wird aktiviert, sobald der Client den Login Button betätigt. Dieser Loading Screen besteht aus einem animierten Gif und einem Hintergrund, welcher dieselbe Farbe hat wie der Hintergrund des Gifs. Somit hat der Client dann das Gefühl, dass er sich erfolgreich eingeloggt hat und nun noch auf den Desktop warten muss. Dieser Loading Screen (Abbildung 34) wird dann so lange ausgeführt, bis das Programm abgebrochen wird.

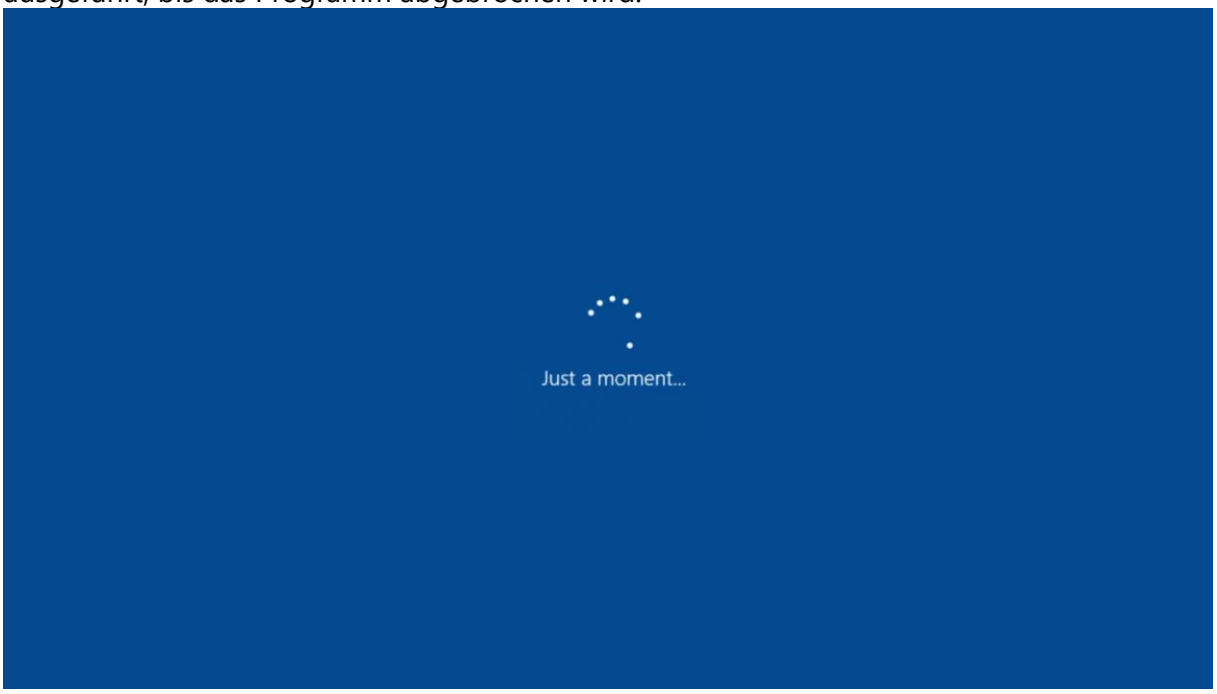


Abbildung 34: Loading Screen (Eigene Darstellung)

► Der komplette Code des Fake-Login-Screens befindet sich im Anhang F.

7.1.2 Einbindung in PyRDP

Damit beim Verbindungsaufbau vom Client zum Server auch zuerst der Fake-Login-Screen angezeigt wird, muss dieser noch in PyRDP eingebunden werden.
Wir haben das mit einem neuen Parameter gelöst. Wenn man einen Man-in-the-Middle Angriff auf einen Server starten will, der nur mit NLA gesicherte Verbindungen akzeptiert, dann muss man beim starten des PyRDP den Parameter «--nla» mitgeben. Als Argument gibt man die IP-Adresse des Servers an, auf welchem der Fake-Login-Screen läuft. Ebenfalls muss man den Username und das Passwort des Servers auf welchem der Fake-Login-Screen läuft, angeben, da man sich sonst auf dem Server selbst anmeldet und danach erst auf den Fake-Login-Screen kommt. Und da das Passwort vom Client sehr wahrscheinlich anders ist als das Passwort für den Fake-Login-Screen Server, würde der Client beim Login Screen des Servers hängen bleiben.
Der Code im PyRDP wurde so verändert, dass wenn der Parameter «--nla» mitgegeben wird, man auf dem Fake-Login-Screen Server weitergeleitet wird und nicht auf den eigentlichen Zielserver.
Wenn man nun auf dem Fake-Login-Screen Server angemeldet ist und den gefälschten Anmeldebildschirm sieht, wird die Variable loggedIn im DeviceRedirection.py File nicht auf True gesetzt und somit zeichnet PyRDP alle Tastaturanschläge bis zu einem «Return» Tastenanschlag auf. Nachdem «Return» gedrückt wurde, werden die Tastaturanschläge in einer Variablen gespeichert. Danach wird dieser String noch beim Tabulatorzeichen ('\t') getrennt und man hat den Usernamen und das Passwort, und wird mit der CredSSP Authentifizierung fortgefahren.
Nachdem die Credentials gespeichert wurden, wird die Variable loggedIn im BasePathMITM.py File auf True gesetzt, damit PyRDP nicht die ganze Zeit meint, der Benutzer möchte sich anmelden und alle Tastaturanschläge aufzeichnet und nach jedem Return diese Tastaturanschläge abspeichert.
Ein Beispiel wie man PyRDP mit dem Fake-Login-Screen startet, wäre folgendes:

```
pyrdp-mitm.py 10.1.1.1 --username user1 --password securepassword --nla 10.1.2.3
```

Wobei die IP-Adresse 10.1.1.1 der eigentliche Zielserver ist, user1 der Username vom Server auf welchem der Fake-Login-Screen läuft, «securepassword» das Passwort für diesen Server und 10.1.2.3 die IP-Adresse des Servers auf welchem der Fake-Login-Screen läuft.

7.2 **CredSSP Authentifizierung**

Das CredSSP-Authentifizierungsverfahren ist bereits in vielen Python-Implementierungen vorhanden. Daher musste dieser Teil unserer MITM-Variante nicht komplett von Grund auf neu implementiert werden. Es gibt bekannte Bibliotheken wie etwa impacket oder pywinrm, die sich in bestehende Anwendungen einfach einbinden lassen. Es gibt auch bereits einige RDP-MITM-Tools, die auf Python basieren und dabei diese Bibliotheken verwendet haben.
Daher wurde die Implementierung der CredSSP-Authentifizierung nicht von Grund auf neu geschrieben, sondern die impacket-Bibliothek wurde für diesen Zweck verwendet. Die Bibliothek konnte auch in PyRDP ähnlich wie die anderen Tools integriert werden, allerdings waren einige kleinere Änderungen notwendig.

Für die Implementierung wurde die CredSSP-Authentifizierungsimplementierung aus dem Tool CredSSPY übernommen und angepasst. Zunächst musste die Einbeziehung der Domain in die Challenge-Berechnung entfernt werden, da CredSSPY einen anderen MITM-Ansatz verwendet. Dies konnte problemlos umgeschrieben bzw. aus der Berechnung entfernt werden, da in den NTLM-Nachrichten die Angabe der Domain ein optionales Feld ist.

Der Authentifizierungsprozess wird mit einer neuen, in PyRDP implementierten Methode namens «clientConnect» gestartet und umfasst fünf Schritte:

1. TLS-Verbindung mit Server aufbauen (→ TLS-Handshake)
2. Starten des CredSSP-Handshakes
   2.1. Senden der NTLM NEGOTIATE_MESSAGE an Server (SPNEGO Token)
   2.2. Empfangen der NTLM CHALLENGE_MESSAGE von Server (SPNEGO Token)
3. Senden der NTLM Authenticate-Nachricht an den Server (SPNEGO Token)
   3.1. Berechnung der Challenge
   3.2. Signieren des Server Public Keys (Feld «pubKeyAuth»)
4. Überprüfung der Public Key-Signatur des Servers
5. Senden der verschlüsselten Anmeldeinformationen an den Server

Die Methode clientConnect wird in der bestehenden Klasse BasePathMITM aufgerufen, sobald die Anmeldeinformationen (Benutzername und Kennwort) über den gefälschten Anmeldebildschirm erfasst worden sind.

Um die erfolgreiche Authentifizierung über NLA (CredSSP) des PyRDP-Clients zum RDP-Server zu überprüfen, wurde das Tool Event Viewer von Microsoft verwendet, das standardmässig im Windows-Betriebssystem enthalten ist. Im Event Viewer wird ein Protokoll der Anwendungs- und Systemmeldungen, einschliesslich Systemanmeldungen von Remote-Benutzern, angezeigt.
Mit den Security-Logs (Abbildung 35) konnte sichergestellt werden, dass die Implementierung der CredSSP-Authentifizierung erfolgreich umgesetzt wurde.

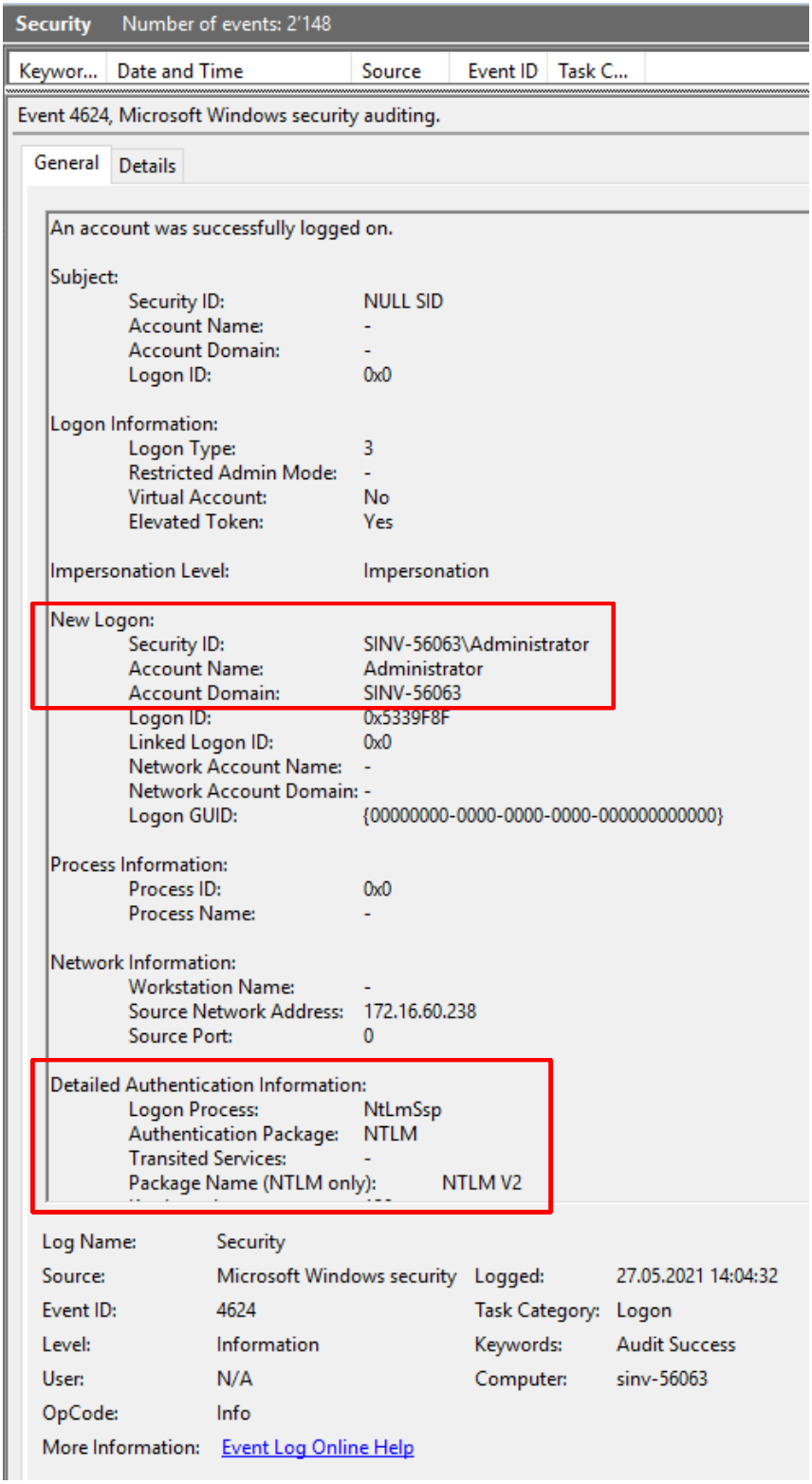


Abbildung 35: Event Viewer - CredSSP Auth Verifizierung (Eigene Darstellung)

► Der komplette Code der CredSSP Authentifizierung befindet sich im Anhang G.

7.3 **Layer Mapping**

Für das Layer Mapping von RDP wurden die folgenden drei verschiedenen Implementationsansätze verfolgt:

- Neue RDPMITM Instanz
- Ganze App neustarten nach dem Fake-Login Screen
- Neuer Observer / Layer / Parser

Ansatz: Neuer RDPMITM Instanz

Die Klasse RDPMITM.py ist die Hauptklasse des mitm-Moduls. Sie verbindet sich mit dem Server und dem Client und baut die verschiedenen Kanäle und Layers von RDP auf einer TCP Verbindung auf. Die Kanäle zum Server und zum Client werden dann auch miteinander verbunden, so dass die Daten von Client zum Server und umgekehrt weitergeleitet und auch ausgelesen werden können.
Wir haben versucht, diese Klasse nach dem Einlesen der Credentials via Fake-Login-Screen in BasePathMITM.py neu zu instanziieren, damit die ganzen Layers und Kanäle mit der Zielserver-Verbindung aufgebaut werden. Nachdem die Credentials eingelesen wurden, wurde eine TLS Verbindung erfolgreich aufgebaut und wir konnten uns auf dem Server erfolgreich authentifizieren, mit den Credentials, die wir vom Fake-Login-Screen haben. Nach mehreren Versuchen wurde jedoch die Fehlermeldung «object is not callable» ausgegeben. Diese Fehlermeldung konnte behoben werden. Jedoch konnten wir unsere bereits aufgebaute TLS Verbindung zum Server nicht in den RDPMITM einbinden, so dass die RDP Layers aufgebaut wurden und deshalb wurde dieser Ansatz später verworfen.

Ansatz: Ganze App neustarten nach dem Fake-Login-Screen

Der PyRDP-Man-in-the-Middle wird mit dem Skript PyRDP-mitm.py gestartet. Wir haben versucht, ähnlichen Code aus PyRDP-mitm.py zu übernehmen und den Reaktor von der Twisted Library zu stoppen und erneut einen Reactor zu starten. Dieser Reactor ist der Event Loop in der Twisted Library. Über ihn laufen alle eingehenden und ausgehenden Nachrichten.
Nach dem Auslesen der Credentials über einen gefälschten Anmeldebildschirm in BasePathMITM.py wurde die CredSSP-Authentifizierung in RDPMITM mit den erhaltenen Credentials durchgeführt. Nach der erfolgreichen CredSSP-Authentifizierung wurde das Layer-Mapping jedoch nicht automatisch über die Observer durchgeführt, da das PyRDP-Projekt die TLS-Verbindung, die vorhin mit CredSSP hergestellt wurde, nicht kannte und somit die Layers nicht auf dieser Verbindung aufbauen konnte. Bei diesem Ansatz konnte diese TLS-Verbindung ebenfalls nicht eingebunden werden. Ebenfalls Probleme bereitete der Reactor. Denn das Problem ist, dass nicht zwei Reactors gleichzeitig auf denselben Port hören können. Deshalb mussten wir bei diesem Ansatz für die Verbindung zum Fake-Login-Screen auf einen anderen Port hören. Ein weiteres Problem war, dass der Reactor nicht gestoppt werden konnte, trotz mehreren Ansätzen. Auch war dieser Ansatz so nicht realistisch, da sich der Client nach der Eingabe der Credentials auf einen neuen Port verbinden müsste. Denn normalerweise versucht sich der Client wieder zu verbinden, wenn die Verbindung Serverseitig unterbrochen wurde, jedoch immer auf dieselbe Adresse und denselben Port. Deshalb wurde auch dieser Ansatz nach mehreren Versuchen verworfen.

Ansatz: Observer / Layer / Parser

Wie schon in der Analyse erwähnt, verwendet das PyRDP Projekt das Observer Pattern für die verschiedenen Layers wie z.B. X224 und MCS. Mithilfe dieser Observer wird auf eine einkommende Nachricht reagiert und entsprechende Aktionen ausgelöst, wie z.B. der Aufbau eines virtuellen Channels. Wir haben für die CredSSP-Authentifizierung einen neuen Observer, ein neuer Layer und einen neuen Parser erstellt. Der neue CredSSP Layer wird nach dem X224 und vor dem MCS Layer im File layerset.py eingebunden und aufgebaut. In diesem CredSSP Layer soll die CredSSP Authentifizierung auf der bereits ausgebauten TLS-Verbindung durchgeführt werden. Nach einer erfolgreichen Authentifizierung sollen die verschiedenen Kanäle, welche PyRDP schon anbietet, via Observer aufgebaut werden. Jedoch wurde der neu implementierte CredSSP Observer nie aufgerufen und deshalb wurde die CredSSP Authentifikation nach der ersten Message nicht weitergeführt. Ebenfalls sendete der Server eine TLS Fehlermeldung nach Erhalt der ersten NTLM Message zurück. (Abbildung 51 Im Anhang I)

Dieser Ansatz ist der meist versprechende Ansatz im Vergleich zu anderen zwei Ansätzen. Jedoch wurde dieser Ansatz nicht vollumfänglich implementiert.

Teil III

Anhang

8 Weitere Themen

8.1 **RDP Historie**

Der Inhalt und die Ausführungen des folgenden Abschnitts beruhen sich auf folgenden Quellen: (Wikipedia, 2021; Bordes, Ebalard, & Rigo, 2021)

8.1.1 Protokoll-Versionen

RDP 4.0 (Windows NT Server 4.0 TSE)
RDP 4.0 ist die erste Version des RDP-Protokolls, die 1996 mit Windows NT Server 4.0 in der Terminal Server Edition eingeführt wurde. Das Protokoll war zu diesem Zeitpunkt in seiner Funktion noch recht eingeschränkt. Das Hauptziel dieses Protokolls bestand darin, eine Lösung für den Display-Offset auf einem Benutzer-Endgerät bereitzustellen und den verschlüsselten Transport von Maus- und Tastatureingaben des Benutzers zu ermöglichen.
Das Protokoll bot auch in der ersten Version schon drei Verschlüsselungsstufen zur Auswahl: niedrig, mittel (Standard) und hoch. In RDP 4.0 basierten diese verschiedenen Stufen alle auf der Verwendung einer 40-Bit-RC4-Verschlüsselung. Die Integrität wurde über ein Pseudo-HMAC 16 sichergestellt. Die in dem Protokoll verwendeten Sicherheitsmechanismen (Authentifizierung, Schlüsselaustausch, etc.) waren zu diesem Zeitpunkt ineffektiv und waren daher anfällig auf Angriffe aus der Umgebung. Ausserdem war es auch möglich, den während einer Sitzung ausgetauschten Datenverkehr zu entschlüsseln. Diese Schwachstellen wurden erst fast 10 Jahre später mit der Version 5.2 des Protokolls behoben.

RDP 5.0 (Windows 2000 Server)
Mit dieser Version, die mit Windows 2000 Server eingeführt wurde, wurden neue Funktionen eingeführt: persistenter lokaler Bitmap-Cache, Wiederaufnahme von Sitzungen, lokales Drucken, gemeinsame Nutzung der Zwischenablage und Netzwerkoptimierungen. Windows 2000 unterstützte eine Schlüsselgrösse von 56 Bit für die RC4-Verschlüsselung.
Durch Installation des Windows 2000 High Encryption Pack oder SP2 konnte aber dann die Verwendung einer 128-Bit-Schlüsselgrösse aktiviert werden. Damit die Vertraulichkeit und Integrität der Verbindung gewährleistet werden konnte, musste in dieser Version des RDP-Protokolls mindestens das SP2 installiert werden. Die Sicherheitslücken aus der vorherigen Version, wurden erst mit dem SP4 behoben.

RDP 5.1 (Windows XP Professional)
Diese Version wurde 2001 mit Windows XP Professional eingeführt. Sie unterstützt 24-Bit-Farbe, Festplattenumleitung, Audio und COM-Ports. Ausserdem bietet diese Version Unterstützung für die Authentifizierung mittels Smartcard.
Ein RDP 5.1-Client war für Windows 2000, Windows 9x und Windows NT 4.0 verfügbar. Mit diesem Release wurde der Name der Client-Software von «Terminal Services Client» in «Remote Desktop Client» geändert.
In dieser Version wurde aus Sicherheitsgründen eine neue Verschlüsselungsstufe, FIPS, zu den drei zuvor eingeführten RC4-basierten Stufen hinzugefügt, die Triple DES für die Verschlüsselung verwendet und HMAC-SHA1 für die Integrität benutzt.

RDP 5.2 (Windows Server 2003 SP1)
Diese Version wurde mit Windows Server 2003 SP1 eingeführt. RDP 5.2 bietet die Möglichkeit, TLS zum Schutz von RDP-Sitzungen zu verwenden. Einer der Vorteile dieser Erweiterung ist die Fähigkeit des Clients, den Server zu authentifizieren. Der Einsatz von TLS für den RDP-Datenverkehr ermöglicht es zudem die Verwendung von optimierten Verfahren für Schlüsselaustausch, Verschlüsselung und die Sicherstellung der Integrität.

RDP 6.0 (Windows Vista)
Diese Version wurde im Jahr 2007 mit Windows Vista eingeführt. RDP 6.0-Client-Versionen sind für Windows Server 2003 und Windows XP SP2 verfügbar. Aus funktionaler Sicht bietet diese Version Unterstützung für den 32-Bit-Farbmodus und die Unterstützung für die Verwendung von mehreren Bildschirmen. Eine der wichtigsten Sicherheitsentwicklungen von RDP 6.0 ist die Integration von NLA. NLA ermöglicht die Authentifizierung des Benutzers zu Beginn der RDP-Sitzung und ermöglicht anschliessend die direkte Übermittlung von dessen Anmeldedaten an den Server.
Bei Windows XP wurde die CredSSP-Unterstützung erst nach einem Upgrade auf SP3, der Aktivierung des CredSSP-Mechanismus und der Installation von Version 6.1 oder höher des RDP-Clients integriert. Parallel wurde mit dieser Version auch ein Microsoft Remote Desktop Connection Client für OS X eingeführt, welches sowohl Intel- wie auch PowerPC-Geräte mit der OS Version 10.4.9 und höher unterstützt.

RDP 6.1 (Windows Server 2008)
Diese Version des RDP Protokolls wurde zusammen mit Windows Vista SP1 und Windows Server 2008 eingeführt. Client-Updates auf diese Version waren für Windows Vista SP1 sowie für Windows XP SP2 verfügbar. Auch in dieser Version wurden neue Funktionen eingeführt: Remote-Verbindung zu einzelnen Programmen, clientseitiges Druckerumleitungssystem, Remote-Administrator, der Software sowie Einstellungen auf dem Client kontrollieren kann.

RDP 7.0 (Windows 7)
Diese Version wurde mit Windows 7 und Windows Server 2008 R2 eingeführt. Diese Version kann zudem als Updates für Windows XP SP3, Windows Vista SP1 und SP2 geladen werden. Aus funktionaler Sicht bietet diese Version insbesondere bidirektionale Audiounterstützung, eine Beschleunigung des Bitmap-Austausches, Aero-Unterstützung, Language Bar Docking, Shadow-Funktion und Unterstützung für einen echten Multi-Screen-Modus.
Um die Sicherheit zu erhöhen, verwendet RDP 7.0 ein selbstsigniertes 2048-Bit RSA-Zertifikat mit einer Gültigkeit von 6 Monaten, das vom System generiert und standardmässig für die TLS-Sitzung verwendet wird.

RDP 7.1 (Windows 7 SP1)
Diese Version wurde zusammen mit Windows 7 SP1 und Windows Server 2008 R2 SP1 eingeführt. Mit dieser Version wurde RemoteFX eingeführt, das virtualisierte GPU-Unterstützung und Host-seitige Verschlüsselung ermöglichte.

RDP 8.0 (Windows 8)
Diese Version wurde im Jahr 2012 mit Windows 8 und Windows Server 2012 eingeführt. Auch mit dieser Version wurden funktionale Erweiterungen eingeführt: Unterstützung von Adaptive Graphics, automatische Auswahl von TCP oder UDP als Transportprotokoll, Multi-Touch-Unterstützung, USB-Umleitung und weitere. Eine neue Funktion, die eingeführt wurde, ist die eingeschränkte Unterstützung für RDP Session Nesting. Diese wird jedoch nur von Windows 8 und Windows Server 2012 unterstützt. RDP 8.0-Client- sowie die Server-Komponente sind für Windows 7 SP1 und Windows Server 2008 R2 SP1 verfügbar. Mit dieser Version wurden die Funktionen Aero-Glass-Remoting und Shadow, welche in der Version 7.0 eingeführt wurden, wieder entfernt. Im RDP 8.0-Client wurde eine Schaltfläche «Connection Quality» in der Verbindungsleiste hinzugefügt, die zusätzliche Informationen über die Verbindung angibt.

RDP 8.1 (Windows 8.1)
Die RDP Version 8.1 wurde zusammen mit Windows 8.1 und Windows Server 2012 RS2 eingeführt. Client-Updates auf diese Version sind für Windows 7 SP1 verfügbar. Jedoch wird dabei nur die Client-Komponente geliefert. Falls auch die Server-Komponente benötigt wird, kann KB2592687-Update vor der Installation des RDP 8.1-Updates installiert werden. Mit RDP 8.1 wurde die in der vorherigen Version entfernte Session Shadowing-Funktion wieder eingeführt. Zusätzlich behebt diese Version auch einige visuelle Probleme mit Microsoft Office 2013 bei der Ausführung als RemoteApp. Diese Version ermöglicht einen eingeschränkten Admin-Modus, bei dem für die Anmeldung nur der Hash des Kennworts benötigt wird, wodurch ein Pass-the-Hash Angriff ermöglicht wird.

RDP 10.0 (Windows 10)
Diese Version wurde mit Windows 10 eingeführt. RDP 10.0 ergänzt die Vorgängerversion um die AutoSize-Zoomfunktion und bietet Verbesserungen bei der Grafikkomprimierung.

8.1.2 Sicherheitslücken

Seit der Einführung von RDP wurden etwa vierzehn referenzierte Sicherheitslücken in den Microsoft-Implementierungen gefunden. Diese werden in diesem Abschnitt aufgeführt und in drei Hauptkategorien unterteilt.

- **Denial of Service**
  - MS01-006: Ungültige RDP-Daten können zum Ausfall des Terminalservers führen. (https://docs.microsoft.com/en-us/security-updates/securitybulletins/2001/ms01-006)
  - MS01-040: Ungültige RDP-Daten können ein Speicherleck in den Terminaldiensten verursachen. (https://docs.microsoft.com/en-us/security-updates/securitybulletins/2001/ms01-040)
  - MS01-052: Ungültige RDP-Daten können zum Ausfall des Terminaldienstes führen. (https://docs.microsoft.com/en-us/security-updates/securitybulletins/2001/ms01-052)
  - MS05-041: Sicherheitslücke im Remote-Desktop-Protokoll kann Denial-of-Service ermöglichen. (https://docs.microsoft.com/en-us/security-updates/securitybulletins/2005/ms05-041)
  - MS11-065: Sicherheitslücke im Remote-Desktop-Protokoll kann Denial-of-Service ermöglichen. (https://docs.microsoft.com/en-us/security-updates/securitybulletins/2011/ms11-065)
- **Code Execution**
  - MS02-046: Buffer Overflow im TSAC-ActiveX-Steuerelement kann Code-Ausführung ermöglichen. (https://docs.microsoft.com/en-us/security-updates/securitybulletins/2002/ms02-046)
  - MS09-044: Sicherheitslücken in der Remotedesktopverbindung können Remotecodeausführung ermöglichen. (https://docs.microsoft.com/en-us/security-updates/securitybulletins/2009/ms09-044)
  - MS11-017: Sicherheitslücke in Remote Desktop Client kann Remotecodeausführung ermöglichen. (https://docs.microsoft.com/en-us/security-updates/securitybulletins/2011/ms11-017)
  - MS11-061: Sicherheitslücke in Remote-Desktop-Webzugriff kann Erhöhung der Berechtigungen ermöglichen. (https://docs.microsoft.com/en-us/security-updates/securitybulletins/2011/ms11-061)
  - MS12-020: Sicherheitslücken in Remote Desktop können Remotecodeausführung ermöglichen. (https://docs.microsoft.com/en-us/security-updates/securitybulletins/2012/ms12-020)
- **Design / Crypto**
  - MS02-051: Kryptografischer Fehler im RDP-Protokoll kann zur Offenlegung von Informationen führen. (https://docs.microsoft.com/en-us/security-updates/securitybulletins/2002/ms02-051)
  - KB275727
  - Forsberg03
  - CVE-2005-1794

Hier wird noch einmal eine grobe tabellarische Übersicht über die oben aufgeführten Sicherheitslücken zusammen mit den betroffenen Protokollversionen und Betriebssystemen dargestellt.

RDP Version	Jahr	Unterstützte Betriebssysteme	Bekannte Sicherheitslücken
4.0	1996	Windows NT Server 4.0	MS01-040, MS01-052
5.0	2000	Windows 2000 Server	MS01-006, MS01-040, MS01-052, MS02-051
5.1	2001	Windows XP Professional, Windows 2000, Windows 9x, Windows NT 4.0	MS01-006, MS01-040, MS01-052, MS02-051
5.2	2003	Windows Server 2003 SP1	MS05-041, MS11-017
6.0	2007	Windows Vista, Windows Server 2003, Windows XP SP2	MS05-041, MS09-044, MS02-051
6.1	2008	Windows Vista SP1, Windows Server 2008, Windows XP SP2	MS05-041, MS09-044, MS11-017, MS02-051
7.0	2009	Windows 7, Windows Server 2008 R2, Windows XP SP3, Windows Vista SP1 & SP2	MS11-065, MS09-044, MS11-017, MS12-020, MS02-051
7.1	2011	Windows 7 SP1, Windows Server 2008 R2 SP1	MS11-061, MS12-020
8.0	2012	Windows 8, Windows Server 2012, Windows 7 SP1, Windows Server 2008 R2 SP1	MS11-061, , MS12-020
8.1	2013	Windows 8.1, Windows Server 2012 RS2, Windows 7 SP1	MS12-020

Tabelle 4: Überblick RDP Historie

8.2 **NTLM Responses & Keys**

Der Inhalt und die Ausführungen des folgenden Abschnitts beruhen sich auf folgenden Quellen: (Glass, 2006; Microsoft, 2021)

8.2.1 Responses

LM Response

Die LM-Response wird von den meisten Clients gesendet. Dieses Schema ist älter als die NTLM-Response und weniger sicher. Neuere Clients unterstützen zwar die NTLM-Response, senden aber aus Kompatibilitätsgründen mit älteren Servern in der Regel beide Antworten. Daher sind die Sicherheitslücken der LM-Response bei vielen Clients, die die NTLM-Response unterstützen, noch vorhanden.

Die LM-Response wird wie folgt berechnet:

1. Das Passwort des Benutzers (als OEM-String) wird in Grossbuchstaben umgewandelt.
2. Dieses Passwort wird mit Nullen auf 14 Bytes aufgefüllt.
3. Dieses «feste» Kennwort wird in zwei 7-Byte-Hälften aufgeteilt.
4. Aus diesen Werten werden zwei DES-Schlüssel erstellt (einer aus jeder 7-Byte-Hälfte).
5. Jeder dieser Schlüssel wird zur DES-Verschlüsselung der konstanten ASCII-Zeichenfolge «KGS!@#$%» verwendet (was zu zwei 8-Byte-Chiffretext-Werten führt).
6. Diese beiden Chiffretext-Werte werden zu einem 16-Byte-Wert verkettet, dem **LM-Hash**.
7. Der 16-Byte-LM-Hash-Wert wird auf 21 Byte aufgefüllt.
8. Dieser Wert wird in drei 7-Byte-Drittel aufgeteilt.
9. Diese Werte werden verwendet, um drei DES-Schlüssel zu erstellen (einen aus jedem 7-Byte-Drittel).
10. Jeder dieser Schlüssel wird zur DES-Verschlüsselung der Challenge aus der Typ-2-Message verwendet (was zu drei 8-Byte-Ciphertext-Werten führt).
11. Diese drei Chiffretext-Werte werden zu einem 24-Byte-Wert verkettet. Dies ist die LM-Response.

Für den Fall, dass das Kennwort des Benutzers länger als 15 Zeichen ist, speichert der Host oder Domänencontroller den LM-Hash für den Benutzer nicht. Die LM-Response kann in diesem Fall nicht zur Authentifizierung des Benutzers verwendet werden. Es wird dennoch eine Antwort generiert und in das LM-Antwortfeld geschrieben, wobei ein 16-Byte-Nullwert (0x00000000000000000000000000000000) als LM-Hash in der Berechnung verwendet wird. Dieser Wert wird vom Ziel ignoriert.

NTLM Response

Die NTLM-Response wird von neueren Clients gesendet. Dieses Schema behebt einige der Schwachstellen der LM-Response, wird aber immer noch als ziemlich schwach angesehen. Ausserdem wird die NTLM-Response fast immer in Verbindung mit der LM-Response gesendet. Die Schwachstellen in diesem Algorithmus können ausgenutzt werden, um das Passwort ohne Berücksichtigung der Gross-/Kleinschreibung zu erhalten, und durch Ausprobieren kann das Passwort mit Berücksichtigung der Gross-/Kleinschreibung ermittelt werden, welches in der NTLM-Antwort verwendet wird.

Die NTLM-Antwort wird wie folgt berechnet:

1. Der MD4-Message-Digest-Algorithmus wird auf das Unicode-Passwort mit gemischter Gross- und Kleinschreibung angewendet. Das Ergebnis ist ein 16-Byte-Wert, der **NTLM-Hash**.
2. Der 16-Byte-NTLM-Hash wird auf 21 Byte aufgefüllt.
3. Dieser Wert wird in drei 7-Byte-Drittel aufgeteilt.
4. Diese Werte werden verwendet, um drei DES-Schlüssel zu erstellen (einen aus jedem 7-Byte-Drittel).
5. Jeder dieser Schlüssel wird zur DES-Verschlüsselung der Challenge aus der Typ-2-Message verwendet (was zu drei 8-Byte-Chiffretext-Werten führt).
6. Diese drei Chiffretext-Werte werden zu einem 24-Byte-Wert verkettet. Dies ist die NTLM-Response.

NTLMv2 Response
Wenn NTLMv2 aktiviert ist, wird die NTLM-Response durch die NTLMv2-Response und die LM-Response wird durch die LMv2-Response ersetzt.

Die NTLMv2-Response wird wie folgt berechnet:
1. Der NTLM-Kennwort-Hash wird ermittelt (wie zuvor beschrieben, ist dies der MD4-Digest des Unicode-Kennwortes in Gross- und Kleinschreibung).
2. Der Unicode-Benutzername in Grossbuchstaben wird mit dem Unicode-Authentifizierungsziel verkettet (der Domänen- oder Servername, der im Feld Zielname der Typ-3-Nachricht angegeben ist).
3. Der HMAC-MD5-Algorithmus für den Nachrichtenauthentifizierungscode wird auf diesen Wert angewendet, wobei der 16-Byte-NTLM-Hash als Schlüssel verwendet wird. Das Ergebnis ist ein 16-Byte-Wert - der NTLMv2-Hash.
4. Ein Datenblock, der als «Blob» bezeichnet wird, wird konstruiert.
5. Die Challenge aus der Typ-2-Nachricht wird mit dem Blob konkateniert. Der HMAC-MD5-Algorithmus zur Nachrichtenauthentifizierung wird auf diesen Wert angewendet, wobei der 16-Byte-NTLMv2-Hash (berechnet in Schritt 2) als Schlüssel verwendet wird. Das Ergebnis ist ein 16-Byte-Ausgabewert.
6. Dieser Wert wird mit dem Blob konkateniert, um die NTLMv2-Response zu bilden.

LMv2 Response
Die LMv2-Antwort wird verwendet, um die Pass-Through-Authentifizierung mit älteren Servern kompatibel zu machen. Ältere Server leiten nur die LM-Antwort weiter und erwarten, dass sie genau 24 Byte lang ist. Die LMv2-Antwort wurde entwickelt, um solchen Servern einen ordnungsgemäßen Betrieb zu ermöglichen.

Die LMv2-Response wird wie folgt berechnet:
1. Der NTLM-Kennwort-Hash wird berechnet (der MD4-Digest des Unicode-Kennworts in Großbuchstaben).
2. Der Unicode-Benutzername in Grossbuchstaben wird mit dem Unicode-Authentifizierungsziel (Domänen- oder Servername) konkateniert, das im Feld «Target Name» (Zielname) der Typ-3-Nachricht angegeben ist. Der HMAC-MD5-Nachrichtenauthentifizierungscode-Algorithmus wird auf diesen Wert angewendet, wobei der 16-Byte-NTLM-Hash als Schlüssel verwendet wird. Das Ergebnis ist ein 16-Byte-Wert, der NTLMv2-Hash.

3. Es wird eine zufällige 8-Byte-Client-Nonce erstellt (dies ist die gleiche Client-Nonce, die im NTLMv2-Blob verwendet wird).
4. Die Challenge aus der Typ-2-Nachricht wird mit der Client-Nonce verkettet. Der HMAC-MD5-Algorithmus zur Nachrichtenauthentifizierung wird auf diesen Wert angewendet, wobei der 16-Byte-NTLMv2-Hash (berechnet in Schritt 2) als Schlüssel verwendet wird. Das Ergebnis ist ein 16-Byte-Ausgabewert.
5. Dieser Wert wird mit der 8-Byte-Client-Nonce konkateniert, um die 24-Byte-LMv2-Antwort zu bilden.

NTLM2 Session Response

Die NTLM2-Session-Response kann in Verbindung mit der NTLM2-Sitzungssicherheit verwendet werden (sie wird mit dem Flag «Negotiate NTLM2 Key» verfügbar gemacht). Dies wird verwendet, um in Umgebungen, die keine vollständige NTLMv2-Authentifizierung unterstützen, einen verbesserten Schutz gegen Angriffe mit vorberechneten Wörterbüchern (insbesondere Rainbow-Table-basierte Angriffe) zu bieten.

Die NTLM2-Sitzungsantwort ersetzt die beiden Antwortfelder LM und NTLM und wird wie folgt berechnet:

1. Es wird eine zufällige 8-Byte-Client-Nonce erstellt.
2. Die Client-Nonce wird auf 24 Bytes mit Nullen aufgefüllt. Dieser Wert wird in das LM-Antwortfeld der Typ-3-Nachricht eingefügt.
3. Die Challenge aus der Typ-2-Nachricht wird mit der 8-Byte-Client-Nonce verkettet, um eine Session-Nonce zu bilden.
4. Der MD5-Message-Digest-Algorithmus wird auf die Sitzungs-Nonce angewendet, was zu einem 16-Byte-Wert führt.
5. Dieser Wert wird auf 8 Byte abgeschnitten, um den NTLM2-Sitzungshash zu bilden.
6. Der NTLM-Kennwort-Hash wird ermittelt (wie oben beschrieben ist dies der MD4-Digest des Unicode-Kennworts mit Gross- und Kleinbuchstaben).
7. Der 16-Byte-NTLM-Hash wird mit Nullen auf 21 Byte aufgefüllt.
8. Dieser Wert wird in drei 7-Byte-Drittel aufgeteilt.
9. Diese Werte werden verwendet, um drei DES-Schlüssel zu erstellen (einen aus jedem 7-Byte-Drittel).
10. Jeder dieser Schlüssel wird verwendet, um den NTLM2-Sitzungshash mit DES zu verschlüsseln (was zu drei 8-Byte-Ciphertext-Werten führt).
11. Diese drei Chiffretext-Werte werden zu einem 24-Byte-Wert verkettet. Dies ist die NTLM2-Sitzungsantwort, die in das NTLM-Antwortfeld der Typ-3-Nachricht geschrieben wird.

Anonymous Response

Die Anonymous Response wird angezeigt, wenn der Client einen anonymen Kontext und keinen echten benutzerbasierten Kontext einrichtet. Dies ist typischerweise der Fall, wenn ein «Platzhalter» für Vorgänge benötigt wird, die keinen authentifizierten Benutzer erfordern. So sind anonyme Verbindungen auch nicht mit einem Konto verbunden.

In einer anonymen Nachricht des Typs 3 setzt der Client das Flag «Negotiate Anonymous». Das NTLM-Antwortfeld wird dabei leer gelassen und das LM-Antwortfeld enthält ein einzelnes Null-Byte («0x00»).

8.2.2 Keys

8.2.2.1 User Session Key

Dies ist der grundlegende Schlüsseltyp, der in der Sitzungssicherheit verwendet wird. Dabei gibt es aber viele Varianten. Die verwendete Ableitungsmethode hängt von den in der NTLM-Typ-3-Nachricht gesendeten Antwort ab. Diese Varianten und ihre Berechnungen werden im Folgenden dargestellt.

LM User Session Key
Wird verwendet, wenn nur die LM-Response bereitgestellt wird (d.h. bei Win9x-Clients). Der LM-Benutzersitzungsschlüssel wird wie folgt abgeleitet:

1. Der 16-Byte-LM-Hash (zuvor berechnet) wird auf 8 Byte abgeschnitten.
2. Dieser wird auf 16 Bytes mit Nullen aufgefüllt. Dieser Wert ist der LM-Benutzersitzungsschlüssel.

NTLM User Session Key
Diese Variante wird verwendet, wenn der Client die NTLM-Response sendet. Die Berechnung des Schlüssels ist recht einfach:

1. Der NTLM-Hash wird ermittelt (der zuvor berechnete MD4-Digest des Unicode-Passworts in Gross- und Kleinschreibung).
2. Der MD4-Message-Digest-Algorithmus wird auf den NTLM-Hash angewandt, was zu einem 16-Byte-Wert führt. Dies ist der NTLM-Benutzer-Sitzungsschlüssel.

LMv2 User Session Key
Wird verwendet, wenn die LMv2-Response gesendet wird (aber nicht die NTLMv2-Antwort). Die Ableitung dieses Schlüssels ist ein wenig komplizierter:

1. Der NTLMv2-Hash wird ermittelt (wie zuvor berechnet).
2. Die LMv2-Client-Nonce wird ermittelt (wird in der LMv2-Response verwendet).
3. Die Challenge aus der Typ-2-Message wird mit der Client-Nonce verkettet. Der HMAC-MD5-Message Authentication Code-Algorithmus wird auf diesen Wert angewendet, wobei der NTLMv2-Hash als Schlüssel verwendet wird, was zu einem 16-Byte-Ausgabewert führt.
4. Der HMAC-MD5-Algorithmus wird auf diesen Wert angewandt, wobei wiederum der NTLMv2-Hash als Schlüssel verwendet wird. Der resultierende 16-Byte-Wert ist der LMv2-Benutzersitzungsschlüssel.

NTLMv2 User Session Key
Wird verwendet, wenn die NTLMv2-Response gesendet wird. Die Berechnung dieses Schlüssels ist dem LMv2 User Session Key sehr ähnlich:

1. Der NTLMv2-Hash wird ermittelt (wie zuvor berechnet).
2. Der NTLMv2-«Blob» wird ermittelt (wie in der NTLMv2-Response verwendet).
3. Die Challenge aus der Typ-2-Message wird mit dem Blob konkateniert. Der HMAC-MD5-Message Authentication Code-Algorithmus wird auf diesen Wert angewendet, wobei der NTLMv2-Hash als Schlüssel verwendet wird, was zu einem 16-Byte-Ausgabewert führt.

4. Der HMAC-MD5-Algorithmus wird auf diesen Wert angewandt, wobei wiederum der NTLMv2-Hash als Schlüssel verwendet wird. Der resultierende 16-Byte-Wert ist der NTLMv2-Benutzersitzungsschlüssel.

NTLM2 Session Response User Session Key
Wird verwendet, wenn die NTLMv1-Authentifizierung mit NTLM2-Sitzungssicherheit verwendet wird. Dieser Schlüssel wird wie folgt aus den NTLM2-Session Response-Informationen abgeleitet:

1. Der NTLM-Benutzersitzungsschlüssel wird, wie zuvor beschrieben, ermittelt.
2. Die Session-Nonce wird ermittelt (wie zuvor beschrieben, ist dies die Verkettung der Typ-2-Challenge und der Nonce aus der NTLM2-Session Response).
3. Der HMAC-MD5-Algorithmus wird auf die Session-Nonce angewendet, wobei der NTLM-Benutzer-Sitzungsschlüssel als Schlüssel verwendet wird. Der resultierende 16-Byte-Wert ist der NTLM2-Session Response-Benutzersitzungsschlüssel.

Null User Session Key
Der Null-Benutzer-Sitzungsschlüssel wird verwendet, wenn eine anonyme Authentifizierung durchgeführt wird. Dieser ist einfach und besteht nur aus 16 Null-Bytes («0x00000000000000000000000000000000»).

8.2.2.2 Lan Manager Session Key

Der Lan-Manager-Sitzungsschlüssel ist eine Alternative zu den Benutzer-Sitzungsschlüsseln, der zur Ableitung von Schlüsseln beim NTLM1-Signieren und -Verschlüsseln verwendet wird, wenn das NTLM-Flag «Negotiate Lan Manager Key» gesetzt ist.

Die Berechnung des Lan-Manager-Sitzungsschlüssels läuft wie folgt:
1. Der 16-Byte-LM-Hash (zuvor berechnet) wird auf 8 Byte abgeschnitten.
2. Dieser wird auf 14 Byte mit dem Wert «0xbdbdbdbdbdbdbd» aufgefüllt.
3. Dieser Wert wird in zwei 7-Byte-Hälften aufgeteilt.
4. Aus diesen Werten werden zwei DES-Schlüssel erstellt (einer aus jeder 7-Byte-Hälfte).
5. Jeder dieser Schlüssel wird zur DES-Verschlüsselung der ersten 8 Bytes der LM-Antwort verwendet (was zu zwei 8-Byte-Chiffretext-Werten führt).
6. Diese beiden Chiffretext-Werte werden zu einem 16-Byte-Wert verkettet, dem Lan Manager-Sitzungsschlüssel.

8.3 **Weitere RDP MitM Tools**

Da wir durch die Recherche herausgefunden haben, dass es bereits mehrere Tools für die Durchführung eines Man-in-the-Middle Angriffes auf RDP Enhanced Security existieren. Deshalb wollen wir diese betrachten und ihre Funktionalität kurz beschreiben. Diese Implementierungen können uns allenfalls dabei helfen, unsere Implementierung durchzuführen.

8.3.1 SETH

SETH ermöglicht es, eine RDP Enhanced Security NLA-Verbindung auf RDP Enhanced Security TLS herunterzustufen. Dazu blockiert das Tool Kerberos-Anfragen des Clients und meldet dann, dass der Domaincontroller (DC) nicht verfügbar ist. Anschließend legt es dem Client ein selbstsigniertes und damit für den Benutzer nicht überprüfbares Zertifikat vor, um die Anmeldedaten des Opfers abzufangen, falls dieses die Verbindung trotz der Sicherheitswarnung akzeptiert. Es gibt derzeit keine Möglichkeit, die Verwendung der clientseitigen RDP Enhanced Security NLA zu erzwingen, sodass dieser Angriff weiterhin möglich ist. Es ist jedoch möglich, die Validierung der Server-Authentifizierung zu erzwingen, indem die Authentifizierung über Kerberos oder die Verwendung eines von einer vertrauenswürdigen Stelle ausgestellten Zertifikats erzwungen wird. (GitHub, 2021)

Link zu Projekt: https://github.com/SySS-Research/Seth

8.3.2 CredSSPY

CredSSPY ist ein Tool, mit dem sich Man-in-the-Middle-Angriffsszenarien auf NLA-Verbindungen einrichten lässt. Es führt ein CredSSP Handshake durch und stellt somit einen geschützten TLS Kanal her, welches für einen Man-in-the-Middle Angriff verwendet werden kann. (Bourguenolle & Bertoli, 2021)

Link zum Projekt: https://github.com/croustibaie/CredSSPY

8.3.3 RDPY

RDPY ist eine in Python geschriebene Implementierung des RDP-Protokolls. Sie umfasst sowohl die Server- als auch die Client-Seite des Protokolls. Wie PyRDP ist auch RDPY mit der ereignisgesteuerten Netzwerk-Engine Twisted aufgebaut. RDPY unterstützt sowohl die Standard-Sicherheitsstufe als auch die erweiterte Sicherheitsstufe von RDP. In der erweiterten Sicherheitsstufe ist die Authentifizierung über TLS sowie NLA (CredSSP über NTLMv2) möglich. (GitHub, 2021)

RDPY stellt die folgenden RDP- und VNC-Binärdateien zur Verfügung:

- RDP Man In The Middle Proxy, der Sitzungen aufzeichnet
- RDP-Honeypot
- RDP Screenshoter
- RDP-Client
- VNC-Client
- VNC-Screenshoter
- RSS-Player

Link zum Projekt: https://github.com/citronneur/rdpy

9 Installation & Handhabung

9.1 **PyRDP-Client Installation**

PyRDP lässt sich mittels Docker-Image und Git-Source installieren. Eine ausführliche Anleitung für die Installation ist auf der GitHub-Seite von PyRDP vorhanden:
https://github.com/GoSecure/pyrdp#installing

PyRDP setzt Python 3.6 oder höher voraus. Dieses Tool wurde vom Entwickler auf Python 3.6 unter Linux (Ubuntu 18.04) und Windows getestet. Auf OS X wurde es hingegen nicht getestet. (Im Rahmen dieser Arbeit wurde PyRDP auf einer virtuellen Ubuntu 18.04-Maschine mit VirtualBox ausgeführt.)

- VirtualBox – Download: https://www.virtualbox.org/wiki/Downloads
- Ubuntu ISO – Download: https://ubuntu.com/download/desktop

Es wird empfohlen, PyRDP in einer virtuellen Umgebung zu installieren, um Probleme mit Abhängigkeiten zu vermeiden.

9.1.1 Erweiterte PyRDP-Client Installation

Den erweiterten PyRDP-Code mit CredSSP-Authentifizierung und Umgang mit Fake-Login-Screen findet man unter dem folgenden Link:
https://gitlab.ost.ch/ba/rdp-man-in-the-middle/-/tree/master/code/pyrdp

Für das Starten der Alternative Man-in-the-Middle Variante müssen RDP Server und Fake-Login-Screen Server korrekt aufgesetzt werden. (siehe Kapitel 9.2

RDP-Client/-Server Konfiguration & 9.3 Fake Login-Screen Server)

Nach dem Aufsetzen von dem Fake-Login-Screen Server und RDP Server kann das PyRDP-mitm.py File mit den Parametern Zielserver-IP, Username von Fake Server, Passwort von Server und Fake Server IP gestartet werden (Beispielbefehl in 7.1.2 Einbindung in PyRDP):

```
pyrdp-mitm.py <Zielserver-IP> --username <Username> --password <Passwort> --nla <Fake Server-IP>
```

9.2 **RDP-Client/-Server Konfiguration**

9.2.1 Server

Als RDP-Server wurde für diese Arbeit ein virtueller Server mit dem Betriebssystem «Windows Server 2019» verwendet. Prinzipiell kann auch ein beliebiges Windows-Client-Betriebssystem verwendet werden, sofern es die Erzwingung der Authentifizierung auf Netzwerkebene (NLA) für den Fernzugriff unterstützt.

Damit PyRDP in seiner Standard-Funktionalität verwendet werden kann, muss jedoch die Erzwingung von NLA deaktiviert sein. (Siehe: Abbildung 36 & Abbildung 37)

Abbildung 36: Windows Remoteeinstellungen (Eigene Darstellung)

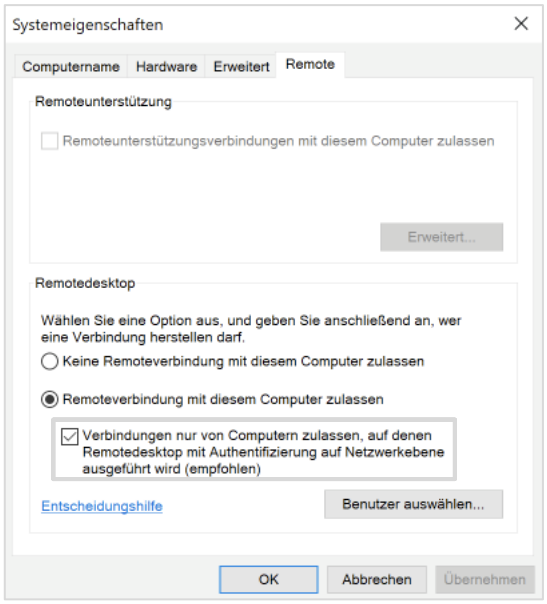


Abbildung 37: Windows NLA aktivieren (Eigene Darstellung)

Um den RDP-Server immer über die gleiche IP-Adresse erreichen zu können, wird empfohlen diesem eine statische IP-Adresse zu vergeben.

9.3 **Fake Login-Screen Server**

Damit PyRDP die Anmeldeinformationen eines RDP-Clients erfassen kann, der versucht eine RDP-Verbindung aufzubauen, muss er die Client-Anfrage an einen Server weiterleiten, der den RDP-Dienst selbst aktiviert hat, damit die Anfrage nicht verworfen wird. Der PyRDP authentifiziert sich jedoch mit einem Konto, das effektiv auf dem Fake-Login-Screen-Server existiert. Nun hat PyRDP die Möglichkeit, die aufgebaute RDP-Verbindung zum Server, auf welchem der Fake Login Screen (7.1 Fake Login Screen) läuft, auf den Client umzuleiten.

Wie jedoch der Fake-Login-Screen auf dem separaten Server gestartet wird, wird im folgenden Abschnitt beschrieben.

1. NLA muss auf dem Fake Login-Screen Server deaktiviert sein. Denn zwischen dem Client und dem PyRDP wird eine RDP Enhanced TLS-Verbindung aufgebaut.
2. Auf dem Fake-Login-Screen Server muss Python installiert werden, damit der Anmeldebildschirm gestartet werden kann.

3. Der im Kapitel 7.1 Fake Login Screen implementierte, gefälschte Windows-Anmeldebildschirm muss auf dem Fake Login Screen-Server gestartet werden.

    a. Dazu muss als erstes der Source-Code des Fake Login Screens lokal abgelegt werden. (Link zum Source-Code: https://gitlab.ost.ch/ba/rdp-man-in-the-middle/-/tree/master/code/fakeLoginScreen)

    b. Danach kann der Fake Login Screen einfach mit dem folgenden Befehl über die Kommandozeile oder einem Python-Editor gestartet werden.

```
python fakeLoginScreen.py
```

9.4 **Wireshark - CredSSP Paketanalyse**

Für die Analyse des RDP-Traffics, mit Wireshark, zwischen Client und Server, der mit NLA/CredSSP geschützt ist, wurden die folgenden Schritte durchgeführt:

Da die RDP-Verbindung mit TLS verschlüsselt wird, muss der private Schlüssel bekannt sein. Daher wurden zunächst ein selbstsigniertes Zertifikat mit dem Format PFX (hauptsächlich in der Windows-Plattform verwendet) und ein privater Schlüssel mit dem Tool OpenSSL erzeugt. Die folgenden OpenSSL-Befehle wurden zur Erstellung des Zertifikats verwendet: (https://adfinis.com/blog/openssl-x509-certificates/)

Schlüsselpaar generieren:

openssl genrsa -out certificate.key 4096

Öffentliches Schlüsselpaar extrahieren:

openssl rsa -in certificate.key -pubout -out certificate_public.key

CSR Datei generieren:

openssl req -new -key certificate.key -out certificate.csr

Selbstsigniertes Zertifikat generieren:

openssl req -text -in certificate.csr -noout -verify

openssl x509 -in certificate.csr -out certificate.crt -req -signkey certificate.key -days 365

crt-Format in pfx-Format umwandeln

openssl pkcs12 -export -out certificate.pfx -inkey certificate.key -in certificate.crt

Anschliessend wurde das Zertifikat mit PFX Format auf dem RDP Server importiert (Abbildung 38). Für Detail siehe «Import the Certificate" auf: https://support.globalsign.com/ssl/ssl-certificates-installation/import-and-export-certificate-microsoft-windows)

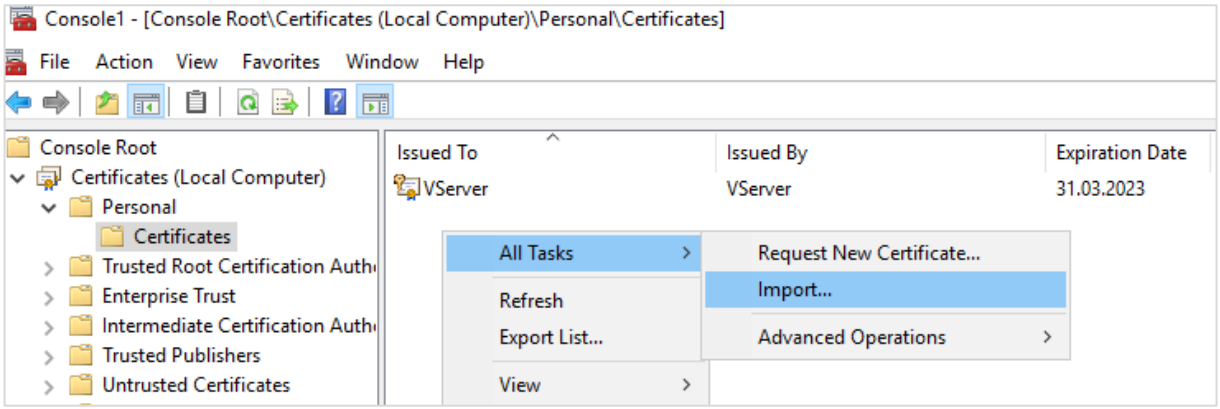


Abbildung 38: Windows Zertifikat importieren (Eigene Darstellung)

Das Zertifikat für RDP-Verbindungen wurde auf dem RDP-Server so konfiguriert, dass das neu importierte Zertifikat für eingehende RDP-Verbindungen mit dem Server verwendet wird. Das wurde gemacht, damit in Wireshark der RDP-Verkehr mit dem bekannten privaten Schlüssel entschlüsselt werden kann. (Siehe https://docs.microsoft.com/en-us/troubleshoot/windows-server/remote/remote-desktop-listener-certificate-configurations)

Ausserdem musste die Perfect Forward Secrecy berücksichtigt werden, da solche Chiffren mehrere Sitzungsschlüssel für eine SSL/TSL-Verbindung erzeugen und nicht mit einem privaten Schlüssel entschlüsselt werden können. Daher wurde auf dem RDP-Server die Verschlüsselungskonfiguration (Abbildung 39) so angepasst, dass die Forward Secrecy nicht unterstützt wird. (Siehe **Step 2** https://unit42.paloaltonetworks.com/wireshark-tutorial-decrypting-rdp-traffic/)

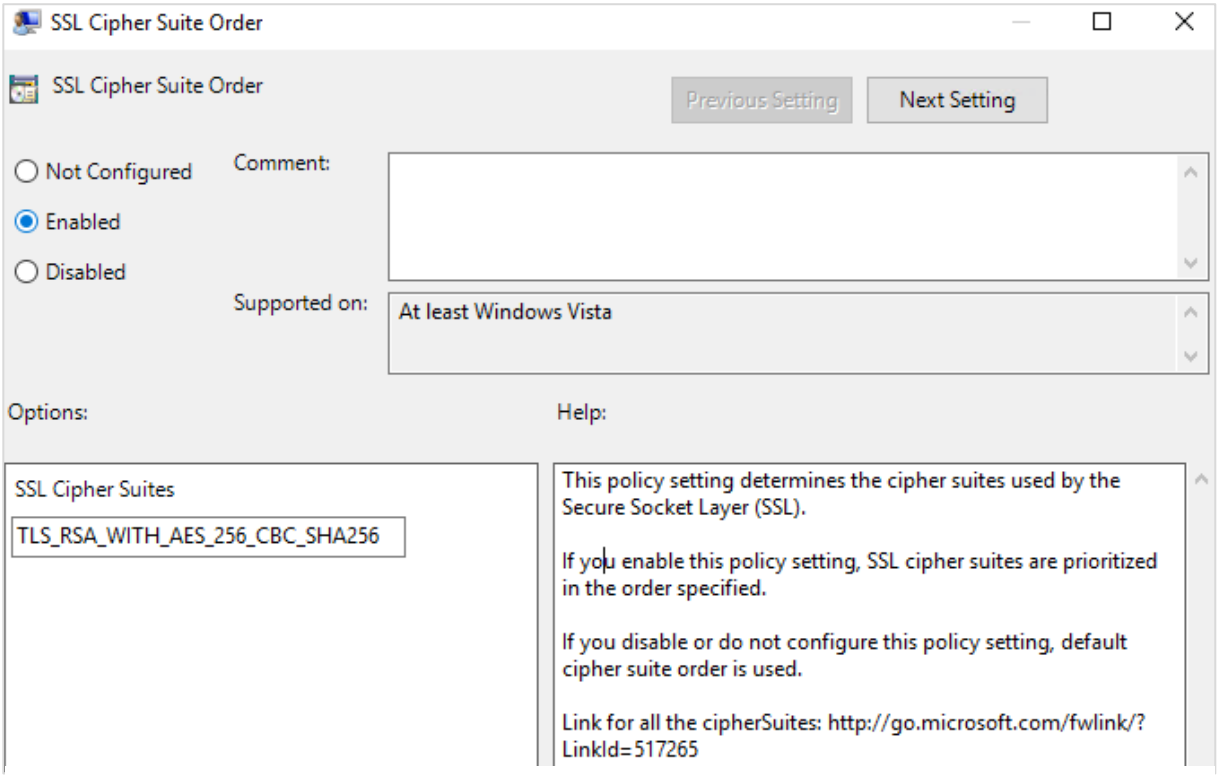


Abbildung 39: SSL Cipher Suite (Eigene Darstellung)

Anschliessend kann der im ersten Schritt erzeugte private Schlüssel in Wireshark unter Preferences > Protocols > TLS > «RSA keys list» > «Edit» angegeben werden (Abbildung 40).

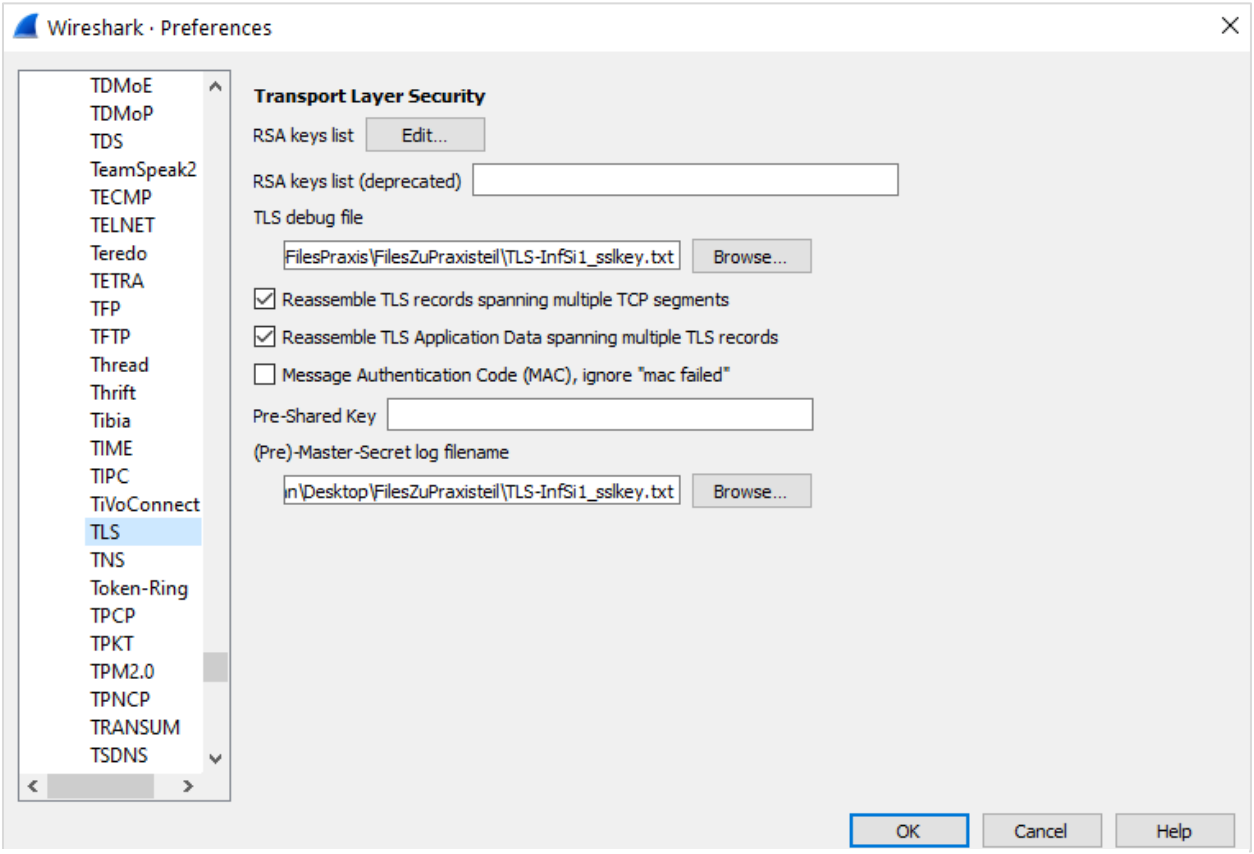


Abbildung 40: Wireshark TLS Einstellungen (Eigene Darstellung)

Der Schlüssel kann wie folgt angegeben werden. (IP address = IP-Adresse vom Server, Port = Port von RDP, Protocol = tpkt, Key File = Datei mit privatem Schlüssel, siehe Abbildung 41)

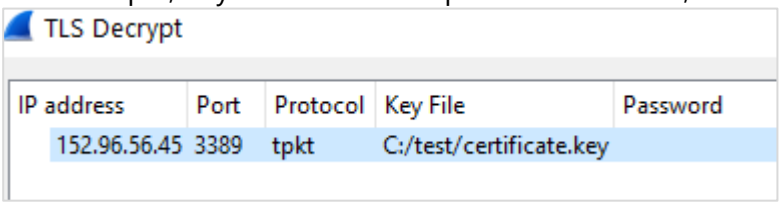


Abbildung 41: TLS Decrypt (Eigene Darstellung)

Nach der Konfiguration des privaten Schlüssels kann die RDP Verbindung TLS-entschlüsselt analysiert werden.

1. Wireshark öffnen und Paket-Erfassung starten. (siehe Abbildung 42)

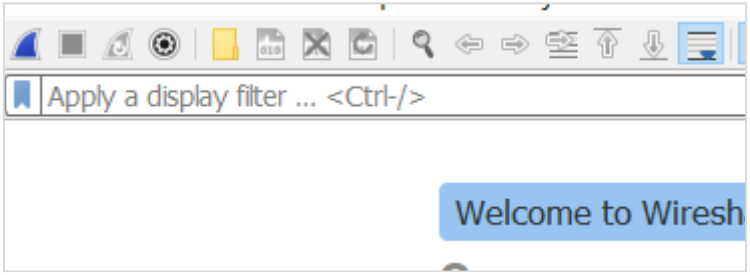


Abbildung 42: Wireshark Packeterfassung (Eigene Darstellung)

2. RDP-Client starten und eine Verbindung zu dem RDP-Server herstellen, auf dem NLA aktiviert ist. (siehe Abbildung 43)

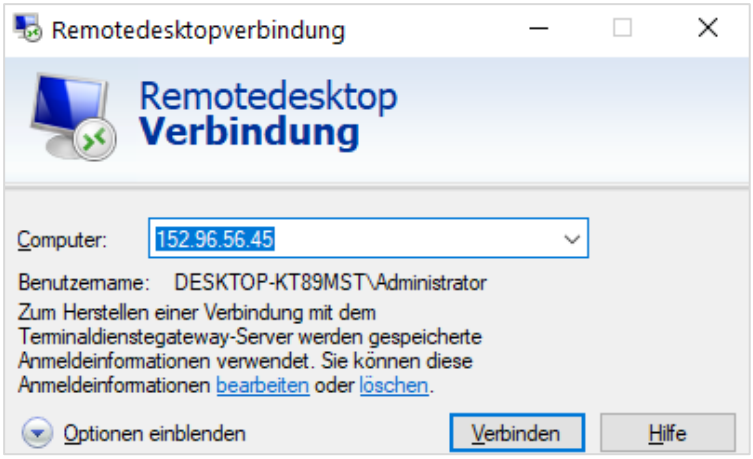


Abbildung 43: RDP Verbindung herstellen (Eigene Darstellung)

Später können alle Pakete zwischen RDP Client und RDP Server anhand der Ziel-IP-Adresse (ip.addr == «IP Adresse Server») gefiltert werden und entschlüsselt angesehen werden.

10 Projektmanagement

10.1 **Einführung**

10.1.1 Zweck

Der Zweck dieses Dokumentes setzt sich aus mehreren Aspekten zusammen.
Wir möchten das komplexe Vorhaben der Bachelorarbeit in mehrere Teilschritte aufgliedern und übersichtlich gestalten. Arbeiten sollen nicht doppelt ausgeführt werden, was wir mit einer präzisen Aufteilung der Aufgaben sicherstellen. Wir möchten realistische Ziele setzen und deren Aufwand und die benötigten Ressourcen korrekt kalkulieren.

10.1.2 Gültigkeitsbereich

Der Gültigkeitsbereich beschränkt sich auf die Projektdauer des Moduls «Bachelorarbeit», im Frühlingssemester 2021. Das Dokument wird von der OST intern verwendet.

10.2 **Projektübersicht**

10.2.1 Ausgangslage

Im Oktober 2020 wurde behauptet, dass PyRDP auch RDP-Verbindungen, die mit NLA/CredSSP geschützt sind, intercepten kann. Jedoch wird erwähnt, dass ein Angriff mittels PyRDP nur möglich, wenn der Angreifer bereits Schlüsselmaterial vom Serversystem gestohlen hat. Ein MitM Angriff könnte aber trotzdem möglich sein. Wir wollen es deshalb genau wissen und das Protokoll im Rahmen der Arbeit genauer analysieren und falls möglich ein PoC implementieren.

10.2.2 Projektcharakter

Das Projekt bzw. die Arbeit unterliegt zeitlichen und personellen Beschränkungen. Bei dieser Arbeit handelt es sich weitgehend um die Analyse der gestellten Hauptaufgabe. Daher ist die Machbarkeit der Aufgabe nicht von Anfang an gewährleistet. Wenn die Analyse die Machbarkeit bestätigt, wird ein Proof of Concept für ein vorgegebenes Tool implementiert. Daher kann diese Arbeit als ein Innovationsprojekt angesehen werden.

10.2.3 Zweck und Ziel

Ziel unseres Projektes bzw. unserer Arbeit ist es, dass wir das bisher im Studium erlerntes Wissen in einer realen Problemstellung anzuwenden.
Aufgrund des fehlenden Vorwissens über die zu verwendenden Tools (PyRDP, RDP), der Programmiersprache (Python) und dem Umfang der Arbeit, wollen wir während unserer Arbeit agil vorgehen.

10.2.4 Lieferumfang

Unsere Arbeit besteht hauptsächlich aus Dokumentation und allenfalls Code-Erweiterungen.
Folgende Dokumente werden am Ende der Arbeit geliefert:

- Abschlussbericht gemäss Aufgabenstellung
- Sitzungsprotokolle
- Code -Erweiterungen (falls möglich)
- Abstract zusätzlich im Online-Tool der OST erfasst
- Poster in elektronischer & Papier Form
- Eigenständigkeitserklärung
- Präsentation

10.2.5 Annahmen und Einschränkungen

Das Projekt verläuft über ein ganzes Semester. Für das Modul gibt es 12 ECTS, was umgerechnet 360 Stunden Arbeit pro Person entspricht. Unser Team besteht aus drei Personen, aufsummiert also 1080 Stunden Arbeit. Das Frühlingssemester umfasst 14 Arbeitswochen, verteilt auf genau 15 Kalenderwochen. Die endgültige Abgabe der Bachelorarbeit hingegen erfolgt zwei Wochen später, 18.06.2021.

10.3 **Projektorganisation**

Das Projekt verfügt über eine flache Organisationsstruktur. Hierbei werden alle wichtigen Entscheidungen im Projektteam getroffen, wobei jedes Mitglied gleichgestellt ist und gleichermassen am Projekt beteiligt, ist. Das Projektteam besteht aus Kevin Moro, Aynkaran Sundralingam und Danusan Premananthan.

10.3.1 Organisationsstruktur

Zusätzlich zu den unten erwähnten Zuständigkeiten ist jedes Projektmitglied dafür verantwortlich, seine Arbeit zu dokumentieren und die Dokumente auf dem neuesten Stand zu halten.

	Danusan Premananthan danusan.premananthan@ost.ch Diplomand - Datenmanagement, Protokollführung
	Kevin Moro kevin.moro@ost.ch Diplomand - Risikomanagement, Infrastruktur
	Aynkaran Sundralingam aynkaran.sundralingam@ost.ch Diplomand - Projektmanagement

Tabelle 5: Organisationsstruktur (Diplomanden)

10.3.2 Externe Schnittstellen

Folgende externe Schnittstellen wurden für unser Projekt definiert.

	Cyrill Brunschwiler cyrill.brunschwiler@ost.ch Betreuer
	Christoph Frei christoph_frei@swissre.com Experte
	Prof. Frank Koch frank.koch@ost.ch Gegenleser

Tabelle 6: Organisationsstruktur (externe)

10.4 **Management Abläufe**

10.4.1 Eckdaten

Das Projekt läuft während 17 Semesterwochen (22.02.2021 bis 18.06.2021). Während diesem Zeitraum arbeitet jedes Projektmitglied (mindestens) 360 Stunden am Projekt, das entspricht 12 ECTS und aufsummiert ergibt das 1080 Arbeitsstunden über das ganze Projekt verteilt.
Bei 17 Wochen resultiert daraus ein Wochendurchschnitt von 21.2 h/Person. Die 17 Wochen ergeben sich aus den 15 Unterrichtswochen im Frühlingsemester 2020, sowie der ersten zwei Juni Wochen.

Beschreibung	Detail
Projektdauer	17 Wochen
Projektmitglieder	3 Personen
Arbeitsstunden pro Person / Woche	21.2 h
Arbeitsstunden (gesamt)	1080 h
Projektstart	Montag, 22. Februar 2021
Projektende	Freitag, 18. Juni 2021 bis 12:00
Präsentation	Freitag, 18. Juni 2021 16:00 - 20:00 Uhr

Tabelle 7: Projekt-Eckdaten

Der Projektumfang erlaubt es, die vorgesehene Zeit vollumfänglich auszunutzen. Sofern die Analyse uns vorzeitig erkennen lässt, dass unser Vorhaben nicht möglich ist, wird in diesem Fall mit dem Betreuer besprochen wie weitergefahren wird.

10.4.2 Zeitliche Planung

Für das Zeitmanagement und das Verwalten der einzelnen Arbeitspakete wird GitLab verwendet. Damit die Planung nach Abschluss des Projekts mit dem effektiven Zeitaufwand verglichen werden kann, soll jegliche Arbeit auf 5 Minuten genau erfasst werden.

10.4.2.1 Projektphasen

Für die Durchführung der Arbeit wird das Vorgehensmodell «RUP» verwendet. Dies ermöglicht ein agiles, iteratives und auf sich aufbauendes Vorgehen. Das Modell wird idealerweise für komplexe und unklare Projektanforderungen, wie es im Falle dieser Arbeit ist, verwendet. Die Phasen Elaboration und Construction werden in mehreren Iterationen durchgeführt. RUP hilft dabei, stabile und gleichzeitig flexible Lösungen zu bauen. Das RUP-Modell teilt den gesamten Projektverlauf in vier Phasen auf: Inception (Konzeption), Elaboration (Entwurf, aber in dieser Arbeit «Analyse»), Construction (Konstruktion) und Transition (Übergabe). (Abbildung 44)

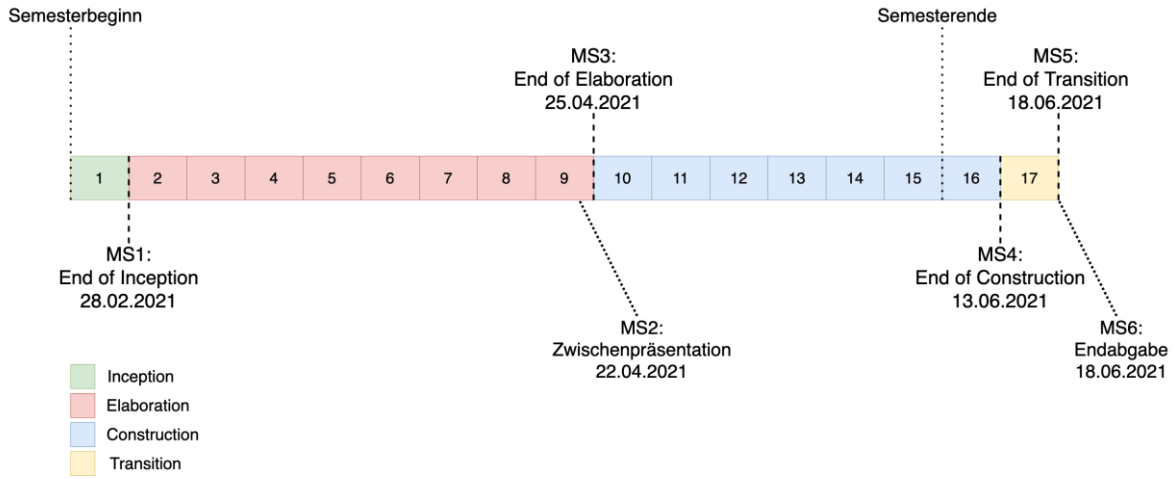


Abbildung 44: Zeitliche Planung (Eigene Darstellung)

Inception (22.02.2021 – 28.02.2021)

Die Hauptaufgabe der Inception Phase ist es, das komplette Projekt, so weit wie es geht, zu planen. Dabei sollen unter anderem die Projekt-Eckwerte wie auch die möglichen Projektrisiken definiert werden. Eine der wichtigsten Aufgaben in dieser Phase ist es, den zeitlichen Ablauf der durchzuführenden Arbeiten zu planen und in Form eines Projektplans festzuhalten. Nebst der Projektplanung sollen auch allgemeine administrative Arbeiten, wie auch die in jedem Fall benötigten Infrastrukturen, wie z.B. GitLab, Datenablage, etc. erledigt bzw. aufgesetzt werden.

Elaboration (01.03.2021 – 25.04.2021)

In der Elaboration Phase geht hauptsächlich um das Einlesen in die relevanten Themen und die Durchführung der Machbarkeitsanalyse für die erhaltene Aufgabenstellung. Dabei sollen die recherchierten Themen, durchgeführten Analysen sowie dabei erhaltenen Erkenntnisse dokumentiert werden. Am Ende der Elaborationsphase geht es um die Beantwortung der Frage, ob ein Man-in-the-Middle-Angriff auf eine mit NLA authentifizierte RDP-Verbindung (CredSSP) möglich ist. Nur falls dies möglich ist, können wir mit der Construction Phase fortfahren. Nebst der Machbarkeitsanalyse soll auch der Code von PyRDP auf Struktur, verwendete Libraries, wichtigste Klassen/Methoden, etc. analysiert und ebenfalls dokumentiert werden.

Construction (26.04.2021 - 13.06.2021)

In der Construction-Phase geht es nun darum, einen Proof of Concept für das PyRDP-Tool zu implementieren, basierend auf der in der vorherigen Phase durchgeführten Analyse. Dabei soll der bereits vorhandene Code um die neue Funktionalität, «Man-in-the-Middle NLA» erweitert werden. Die durchgeführte Implementierung sollte mit sinnvollen Vorgehensweisen und Modellen/Diagrammen, wie sie in den Software-Engineering-Modulen erlernt wurden, dokumentiert werden. Zudem sollen auch die aufgetretenen Probleme und Schwierigkeiten sowie die getroffenen Überlegungen festgehalten werden.

Transition (14.06.2021 – 18.06.2021)

Die Zeit in der Transition-Phase wird genutzt, um den Abschlussbericht und alle notwendigen Dokumente fertig zu stellen und die Präsentation vorzubereiten. Letzte Korrekturen und Verbesserungen sind am Abschlussbericht vorzunehmen. Die Folien für die Abschlusspräsentation müssen vorbereitet werden. Ausserdem müssen in dieser Phase einige administrative Arbeiten erledigt werden, die für die Einreichung der Arbeit notwendig sind.

10.4.2.2 Meilensteine

Folgende Meilensteine werden angestrebt:

ID	Meilenstein	Beschreibung	Termin
MS1	End of Inception	Projektplanung durchführen.	28.02.2021
MS2	Zwischenpräsentation	Zwischenpräsentation durchführen.	22.04.2021
MS3	End of Elaboration	- Theoretische Machbarkeitsanalyse abschliessen. - PyRDP Code analysieren.	25.04.2021
MS4	End of Construction	- PyRDP Code analysieren. - NLA-MitM PoC in PyRDP implementieren.	13.06.2021
MS5	End of Transition	- Dokumentation verbessern und fertigstellen. - Abschlusspräsentation fertigstellen. - Endabgabe vorbereiten.	18.05.2021
MS6	Endabgabe	Endabgabe durchführen.	18.06.2021
MS7	Abschlusspräsentation	Abschlusspräsentation durchführen.	24.06.20201

Tabelle 8: Meilensteine

10.4.3 Besprechungen

Es wurde vereinbart, dass sich alle drei Projektmitglieder mindestens zweimal in der Woche treffen, um sich untereinander auszutauschen, administrative Fragen zu klären und Problematiken zu besprechen. Dazu gehört auch die Durchführung von Reviews und die Definition von den nächsten Arbeitsschritten. Diese Besprechungen werden jeweils jeden Donnerstag und Freitag von 09:00 bis 12:00 Uhr online via Teams abgehalten.
Zusätzlich finden jede Woche Reviews mit unserem Betreuer Herr Cyrill Brunschwiler statt, in welchen jeweils der aktuelle Stand, der zuletzt erreichte Meilenstein sowie die dazugehörenden Arbeitsprodukte bzw. Dokumente, wie auch allfällige entstandene Problematiken besprochen werden.
Alle Besprechungen werden protokolliert und auf GitLab abgelegt.

► Der Inhalt der wöchentlichen Review-Meetings können aus dem GitLab entnommen werden. (https://gitlab.ost.ch/ba/rdp-man-in-the-middle/-/tree/master/minutes)

10.5 **Risikomanagement**

In diesem Abschnitt werden verschiedene Projektrisiken beschrieben, die während des Projektes auftreten können. Zusätzlich wird zu jedem Risiko die Wahrscheinlichkeit des Auftretens und der Gewichtung aufgelistet. Allgemeine Risiken werden hier nicht abgebildet.

Das Ziel ist es, die Risiken so weit wie möglich zu minimieren und falls möglich zu eliminieren.

Die Risiken wurden, während dem Projekt kontinuierlich analysiert und angepasst, um auf mögliche Veränderungen entsprechend reagieren zu können. (Abbildung 45)

10.5.1 Risiken

Diese Matrix symbolisiert den letzten Stand vom 24.02.2021:

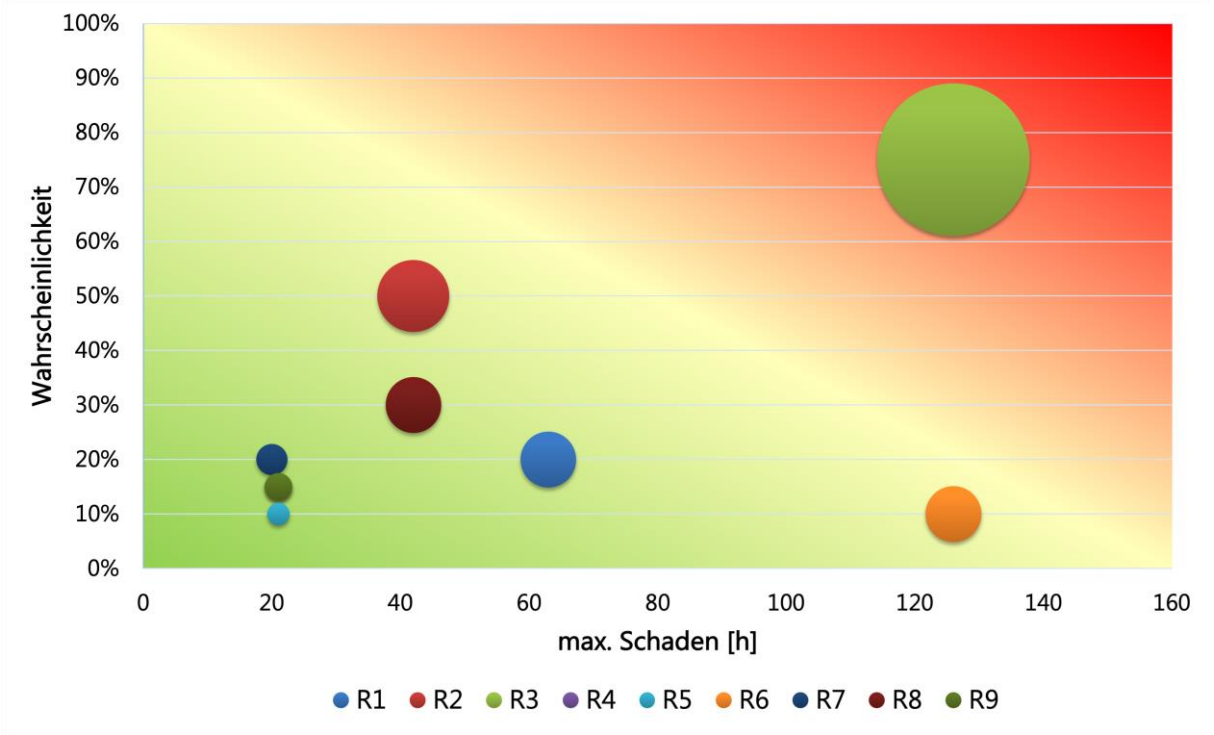


Abbildung 45: Risikomatrix (Eigene Darstellung)

Die Grösse der Blasen entspricht dem gewichteten Schaden
$(Eintrittswahrscheinlichkeit * maximaler\ Schaden)$

ID	Risiko	Eintrittswahrscheinlichkeit	Gewichtung
R1	Kommunikation	20%	12.6
R2	Komplexität	50%	21
R3	Lernkurve	75%	94.5
R4	Ressourcenmangel	15%	23.15
R5	Ausfall externer Komponente	10%	2.1
R6	Umsetzbarkeit der Aufgabenstellung	10%	12.6
R7	Ungenaue Aufwandschätzung	20%	4
R8	Installation & Integration	30%	12.6
R9	Ausfall Personal	15%	3.15

Tabelle 9: Projektrisiken

► Die detaillierte Risikoidentifikation, Analyse und Massnahmen sind im File «Risikoanalyse» beschrieben.

10.5.2 Umgang mit Risiken

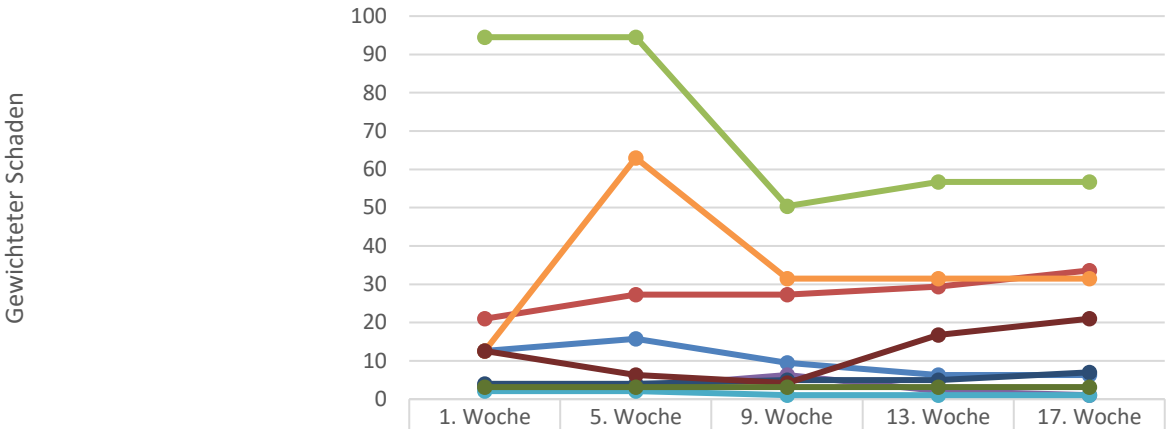
Die Risiken werden so gut wie möglich schon am Anfang des Projektes durch eine gute Projektplanung minimiert.
Für den Fall, dass ein Risiko eintritt, wurden die Risiken, wie sie beschrieben wurden, in die Zeitplanung miteingerechnet. In jedem Arbeitspaket wird daher ein gewisses Zeitpolster miteingerechnet.
Dieses Polster ist 2.5h pro Woche gross. Die Reserven sind somit gleichmässig über alle Projektwochen verteilt.
Falls trotzdem ein beschriebenes Risiko eintritt, werden die Massnahmen, die wir definiert haben, umgesetzt.
Falls es unerwartet viele Probleme im Projekt geben sollte, werden wir die Risikoanalyse neu bewerten und das weitere Vorgehen besprechen.

10.5.3 Risikoentwicklung und Risikoüberwachung

Die Risiken werden im vier Wochen Rhythmus neu beurteilt und eingeschätzt. Somit fliessen die gesammelten Erfahrungen und die neuen Erkenntnisse (insbesondere in Bezug auf die eingesetzten Technologien) in die Matrix ein. Die Schlussbeurteilung erfolgt in einer kurzen Besprechung innerhalb des Teams.

10.5.4 Risikoauswertung

Wie im vorherigen Kapitel definiert, wurde alle vier Wochen eine Risikoanalyse durchgeführt. In der folgenden Grafik (Abbildung 46) sieht man den Verlauf der am Anfang des Projektes festgelegten Risiken.



	1. Woche	5. Woche	9. Woche	13. Woche	17. Woche
R1: Kommunikation	12.6	15.75	9.45	6.3	6.3
R2: Komplexität	21	27.3	27.3	29.4	33.6
R3: Lernkurve	94.5	94.5	50.4	56.7	56.7
R4: Ressourcenmangel	3.15	3.15	6.3	2.1	1.05
R5: Ausfall externer Komponente	2.1	2.1	1.05	1.05	1.05
R6: Umsetzbarkeit der Aufgabenstellung	12.6	63	31.5	31.5	31.5
R7: Ungenaue Aufwandschätzung	4	4	5	5	7
R8: Installation & Integration	12.6	6.3	4.2	16.8	21
R9: Ausfall Personal	3.15	3.15	3.15	3.15	3.15

Abbildung 46: Risikoauswertung (Eigene Darstellung)

Im nächsten Abschnitt werden die aufgetretenen Risken bzw. Schwierigkeiten grob beschrieben.

10.5.5 Aufgetretene Risiken

R1: Kommunikation

Dieses Risiko ist hauptsächlich in der Anfangsphase des Projekts aufgetreten, da zu wenig Vorkenntnisse vorhanden war. Daher kam es vor, dass Arbeiten durchgeführt wurden, die nicht direkt für die Arbeit relevant sind.

R2: Komplexität

Die Arbeit stellte sich als komplex heraus, da wenig Wissen in diesem Thema vorhanden war. Anfänglich fehlte daher auch die Art und Weise wie man an diese Arbeit angehen müsste.

R3: Lernkurve

Da die in dieser Arbeit relevanten Themen sowie das Tool, welches vorgegeben wurde, neu für die Teammitglieder waren, musste für mehrere Wochen mit einer grossen Lernkurve gerechnet werden. In der Elaboration-Phase war das aufgrund der Theorie und die Durchführung der Machbarkeitsanalyse und in der Construction-Phase aufgrund des Tools bzw. dessen Aufbau.

R4: Ressourcenmangel
Aufgrund der aktuellen Situation (Covid-19) und die beschränkte Anzahl von Hardware, welche bezogen werden konnten, musste viel mit virtuellen Maschinen gearbeitet werden. Diese mussten auf den persönlichen Arbeitsgeräten installiert werden, was je nach dem zu Leistungs- und Kompatibilitätsproblemen führte.

R6: Umsetzbarkeit der Aufgabenstellung
Durch die in der Elaboration-Phase durchgeführte Machbarkeitsanalyse konnte festgelegt werden, dass ein Man-in-the-Middle Angriff, wie es in der Aufgabenstellung vorgesehen ist, nicht möglich ist. Daher musste in der zweiten Hälfte der Arbeit die Aufgabenstellung neu definiert werden.

R7: Ungenaue Aufwandschätzung
In der theoretischen Analyse wurde am Anfang oft der entscheidende Faktor nicht genügend tief untersucht, da die Relevanz nicht klar war. Daher mussten gewisse Issues erneut bzw. genauer durchgeführt werden. In der Construction-Phase wurde die Implementierung in drei Teile aufgeteilt, «Fake Login-Implementierung», «NLA Authentifizierung-Implementierung» und «RDP Layer Mapping». Der Aufwand für die «RDP Layer Mapping» wurde unterschätzt, da die Schwierigkeiten nicht von Anfang an klar waren.

R9: Ausfall Personal
In der Semesterwoche vier fiel einer der Teammitglieder aus gesundheitlichen Gründen aus. So geriet die Arbeit für eine kurze Zeit in Verzug, welcher aber daraufhin aufgearbeitet wurde. Während der gesamten Arbeit fielen zwei Review-Meetings, aufgrund das Fehlen des Betreuers, aus. Jedoch hat der Betreuer diese Termine frühzeitig bekannt gegeben. Daher konnte für diese Semesterwochen schon im Vorhinein geplant werden.

10.6 **Arbeitspakete**

Auf Gitlab befinden sich die momentan geplanten Arbeitspakete unter dem Punkt Issues. In der frühen Phase des Projektes werden die Arbeitspakete grob formuliert. Da die Arbeiten erst mit den vorangehenden Prozessen klarer werden, werden wir die Inhalte der Arbeitspakete erst später feiner granulieren.

10.6.1 Auswertung der Arbeitspakete

Für die Auswertung wurden die Arbeitspakete auf die nachfolgenden Labels aufgeteilt:

- Administration & Infrastruktur
- Dokumentation
- Analyse
- Coding
- Team-Meeting
- Review-Meeting

Das Thema RDP Man-in-the-Middle war für das Team ganz neu, deshalb konnten die eingeplanten Meilensteine nicht richtig eingeschätzt und eingehalten werden. Für die Auswertung der Arbeitspakete wurde deshalb nach Labels im Gitlab kategorisiert. (Abbildung 47)

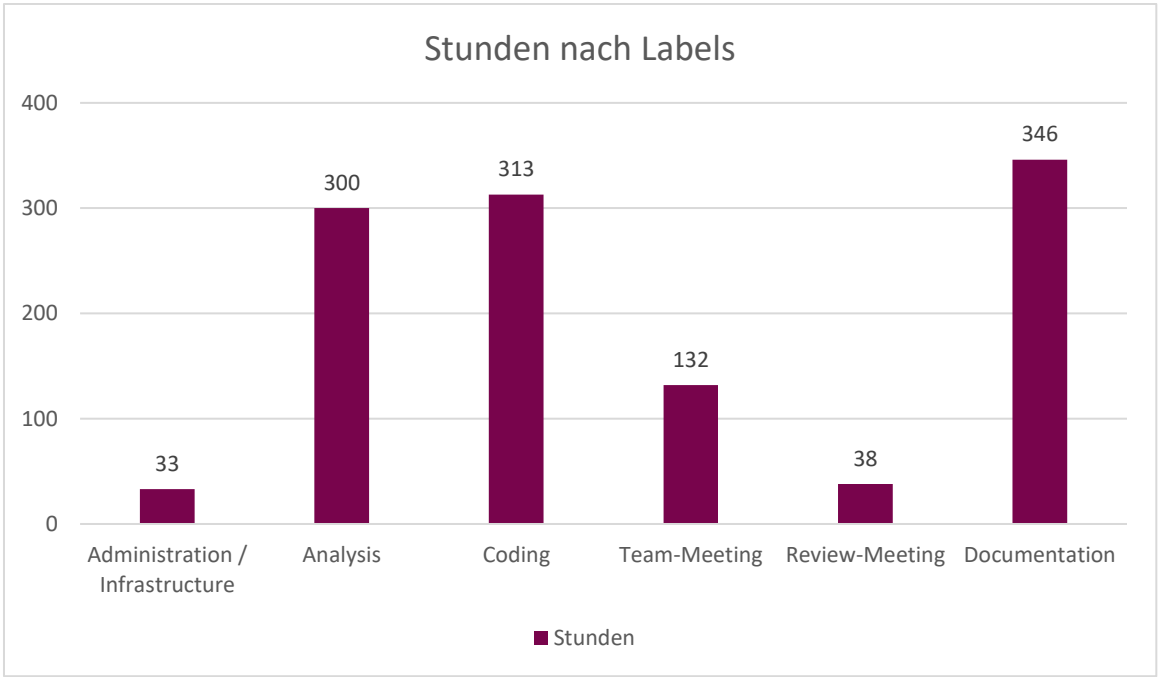


Abbildung 47: Stunden nach Labels (Eigene Darstellung)

Zusätzlich wurde eine Übersicht der Stunden nach Personen erstellt. (Abbildung 48)

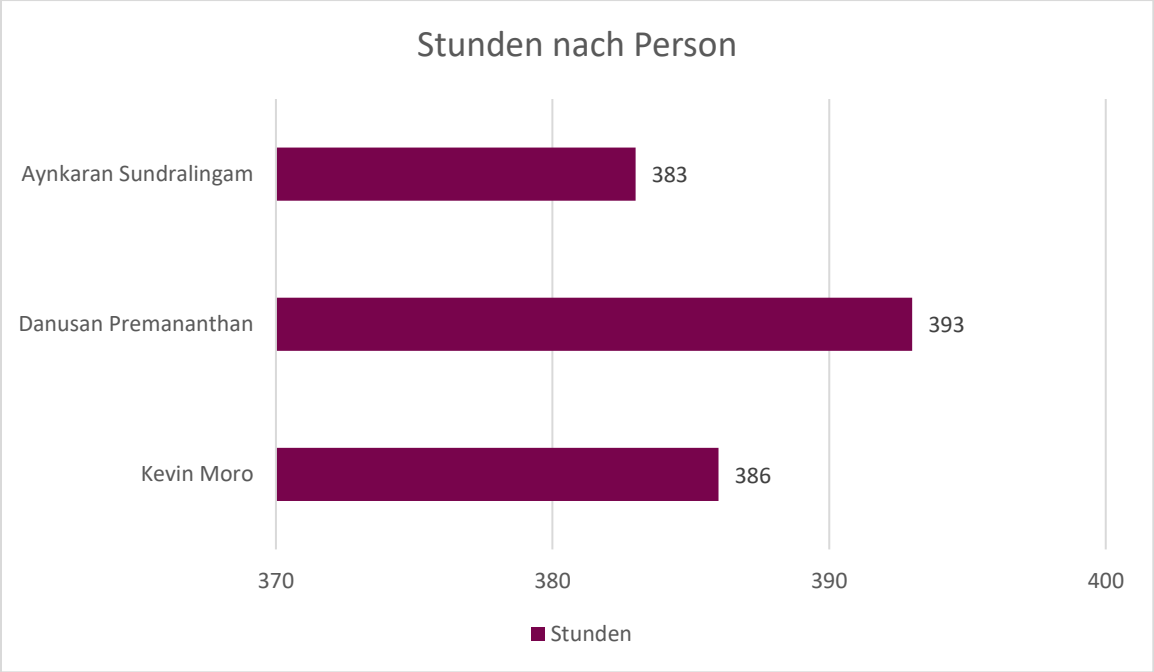


Abbildung 48: Stunden nach Person (Eigene Darstellung)

Insgesamt wurden für diese Arbeit 1162 Stunden benötigt.

10.6.1.1 Administration & Infrastruktur

In diesem Bereich wurden allgemeine Koordinationsaufgaben und Gitlab Aufsetzung in Stunden rapportiert. Da die Teammitglieder mit ihren persönlichen Laptops gearbeitet haben, wurde hier nicht viel Zeit investiert.

10.6.1.2 Dokumentation

In diesem Bereich wurde die allgemeine Dokumentationsaufwand wie z.B. Abstract, Überarbeitung oder Review des Berichts oder Poster eingetragen.

10.6.1.3 Analyse

In dieser Kategorie wurde der Zeitaufwand für die Analyse der Theorie (NLA, CredSSP, NTLM, etc.), den Installationsaufwand und die Codeanalyse des PyRDP-Tools erfasst.

10.6.1.4 Coding

In der Kategorie Coding wurden die Stunden für den Implementierungsaufwand von Fake-Login-Screen, CredSSP-Authentifizierung und Layer-Mapping erfasst.

10.6.1.5 Team-Meeting

In der Kategorie Team-Meeting wurden die Stunden der Meetings rapportiert, welche die Teammitglieder innerhalb des Teams via Microsoft Teams oder am Campus in Rapperswil durchgeführt wurden.

10.6.1.6 Review-Meeting

Bei dieser Kategorie wurde die Zeit der Meetings eingetragen, welche mit dem Betreuer stattgefunden haben.

10.7 **Infrastruktur**

10.7.1 Entwicklungsmaschinen

Die Teammitglieder verwenden ihre persönlichen Arbeitsrechner zur Bearbeitung dieses Projektes. Dabei wird als Betriebssystem Windows 10 sowie macOS 10.14 verwendet.

10.7.2 Virtueller Server

Im Rahmen dieser Arbeit wurden zwei virtuelle Server für die Analyse und Implementierung benötigt. Diese wurden von der OST zur Verfügung gestellt. Aufgrund der aktuellen Situation (Covid-19) konnte nicht aus dem OST-Netzwerk gearbeitet werden. Für den Zugriff auf die virtuellen Server musste VPN verwendet werden.

10.7.3 Übersicht der Tools

Bezeichnung	Verwendungsgrund
GitLab	Handhabung der Arbeitspakete, Zeiterfassung, Code-Ablage, Dokumentenablage
PyCharm, Visual Studio Code	Entwicklungsumgebung (IDE)
Microsoft Office Word	Verfassung von Dokumenten
VirtualBox	Virtualisierungssoftware
Markdown	Verfassung von Protokollen
Dropbox	Dokumentenaustausch
Microsoft Teams	Besprechungen
WhatsApp	Kommunikation innerhalb des Teams

Tabelle 10: Übersicht der verwendeten Tools

10.8 **Qualitätsmassnahmen**

Massnahme	Zeitraum	Ziel der Massnahme
Meeting	Donnerstags und Freitags 09-12 Uhr	Synchronisation: Wo stehen wir? Kommunikation: Teamgeist, Anliegen lösen. Problemlösung: komplizierte technische Probleme gemeinsam angehen.
Pull Request	Vor Merge	GitHub-Flow mit Pull-Request vor jedem Merge.
Code / Dokumentation Review	Wenn nötig	Verständliches Coding, Code-Style Guidelines. Inhaltliche und Sprachliche (Grammatik) Überprüfung.
Review Meeting mit Betreuer	Wöchentlich	Arbeitsstand austauschen, Klärung aufgetauchter Fragen, Frühzeitige Reaktion auf allfällige Unstimmigkeiten oder Missverständnisse.

Tabelle 11: Qualitätsmassnahmen

10.8.1 Dokumentation

Alle Dokumente befinden sich in unserem GitLab- sowie Dropbox-Repository. Das Fileformat ist Markdown, Microsoft Word und PDF. Um das gleichzeitige Arbeiten an den Dokumenten zu vereinfachen und diese zu versionieren, werden diese Dokumente via Dropbox verwaltet. Zusätzlich wird eine Änderungsgeschichte am Anfang des Dokumentes geführt. Die finalen Versionen der Dokumente werden als PDF auf GitLab abgelegt.

► GitLab-Repository: https://gitlab.ost.ch/ba/rdp-man-in-the-middle/

10.8.2 Projektmanagement

Für das Projektmanagement wird GitLab eingesetzt. Administrative Dinge, wie z.B. Zeiterfassung, pflegen wir dort ein.

10.8.3 Merge Requests

Vor jedem Merge eines Branches wird ein Merge Request an einem anderen Peer geschickt. Wenn der Merge überprüft wurde und alles in Ordnung ist, gibt der Peer den Merge frei. Das Issue-Ticket wird dabei erst geschlossen, wenn der Merge durchgelaufen ist oder wenn der Branch aus anderen Gründen gelöscht wurde.

10.8.4 Definition of Done(Dod)

Um Missverständnissen vorzubeugen und eine klare Vorstellung davon zu haben was erwartet wird, wird mit einer «Definition of Done» gearbeitet. Sobald sämtliche vorab definierten Punkte erfüllt sind, kann das Arbeitspaket als erledigt angesehen werden.

DoD für Dokumente:

- Grammatik, Orthografie korrekt
- Wesentliche Punkte abgearbeitet (Ermessens Sache)
- Review durch Betreuer durchgeführt

DoD für Code (falls Coding nötig)

- Merge-Request für Code Review durch ein anderes Teammitglied (Vier-Augen-Prinzip):
  - Code kompiliert einwandfrei
  - Alle Akzeptanzkriterien werden erfüllt
  - Coding Guidelines sind erfüllt
  - Keine kritischen Bugs zu diesem Thema offen.
- Zusammenführung im Master Branch

10.8.5 Dokumentation & Code Style Guidelines

Im Code wird ausschliesslich die englische Sprache benutzt, z.B. für Code Kommentare, Variablennamen, Funktionsnamen usw. Die Dokumentation wird hingegeben in deutscher Sprache gehalten.
Unsere Review Strategie liegt folglich in den Pull-Requests. Wichtige oder heikle Dokumente werden in den Teammeetings abgesegnet. Im Grossen und Ganzen setzt sich alles zusammen aus: Pull Requests, Team-Meetings, Code-Reviews und eventuell noch Remote-Pair-Programming.

11 Reflexion

Aufgrund des grossen Interesses aller drei Diplomanden an Informationssicherheit und Programmierung, haben wir uns für diese Arbeit entschieden und als erste Auswahl beworben. Am Anfang der Arbeit standen wir dennoch vor einer grossen Herausforderung, da keiner der Diplomanden Erfahrung in der Durchführung einer Machbarkeitsanalyse zu einem Informationssicherheitsthema, in unserem Fall «RDP Man-in-the-Middle», und generell in diesem Fachgebiet hatte. Daher war die Vorgehensweise bei dieser Arbeit zunächst unklar. Auch für die Themen, die im Rahmen dieser Arbeit betrachtet wurden bzw. relevant waren, musste das Wissen in den meisten Fällen zuerst noch angeeignet werden.

In der Bachelorarbeit wurde vorausgesetzt, dass zu Beginn der Arbeit ein zeitlicher Ablaufplan erstellt wird, was aufgrund der fehlenden Kenntnisse und Erfahrungen nicht genau durchgeführt werden konnte. So kam es in der ersten Hälfte der Arbeit vor, dass die wöchentlichen Ziele nicht optimal erreicht wurden und eine Nachbearbeitung erforderlich war. Da jedoch für die Durchführung der theoretischen Machbarkeitsanalyse ausreichend Zeit inklusive Puffer eingeplant wurde, konnte diese bis zum geplanten Meilenstein erreicht bzw. abgeschlossen werden. Die Machbarkeitsanalyse zeigte, dass die Durchführung eines direkten (ohne vorhergehende Angriffe) Man-in-the-Middle-Angriffs auf RDP mit der erweiterten Sicherheit «NLA» nicht möglich ist. Dies führte dazu, dass eine der Hauptaufgaben der Arbeit nicht durchgeführt werden konnte. Denn nebst der Machbarkeitsanalyse umfasste die Arbeit auch die Analyse und Implementierung eines Man-in-the-Middle PoC für das Tool «PyRDP».

Da die Bachelorarbeit zusätzlich zum Bericht auch die Lieferung eines Produktes (Hardware, Software, etc.) vorsieht, musste eine alternative Aufgabe gefunden werden. Durch das angeeignete Wissen konnten wir einen anderen Ansatz finden, mit dem die ursprüngliche Aufgabe realisiert werden konnte. Aufgrund dieser «Arbeitsunterbrechungen» haben wir etwa eine Woche Zeit verloren. Zu diesem Zeitpunkt mussten wir das Vorgehen für die nächsten Wochen neu planen und den zeitlichen Aufwand einschätzen. Allerdings waren wir zu diesem Zeitpunkt trotz der Umstände optimistisch, dass es zeitlich funktionieren wird.

Bei der Umsetzung des PoCs für PyRDP stellte sich heraus, dass die Implementierung umfangreicher war, als wir es geschätzt hatten. Zum einen musste der PyRDP-Code detaillierter analysiert werden, zum anderen erwies sich auch die Implementierung als nicht einfach. Die Implementierung wurde in drei einzelne, auf sich aufbauenden Schritten unterteilt. Von diesen drei Schritten konnten auf Grund der zeitlichen Beschränkungen schliesslich nur zwei vollständig implementiert werden. Ansonsten konnten die restlichen Ziele der Aufgabenstellung erreicht werden.

Abgesehen von den aufgetretenen Schwierigkeiten hat die Zusammenarbeit sowohl mit dem Betreuer als auch zwischen den Diplomanden gut funktioniert. Da wir die Arbeitsweise und die Stärken sowie Schwächen des jeweils anderen bereits aus früheren Projekten kannten, hat auch die Aufgabenteilung sehr gut funktioniert.
An dieser Stelle möchten wir uns bei unserem Betreuer Cyrill Brunschwiler für die Unterstützung und angenehme Zusammenarbeit bedanken.

12 Anhang A: Glossar

A	
ASCII	7-Bit-Zeichenkodierung (American Standard Code for Information Inter)
Authenticator	Überprüft die Echtheit von Client
Avatarbild	Ersatzbild für Profile, welche man im Internet benutzt
B	
Bibliothek (Library)	Sammlung von Unterprogrammen in Programmierung
Bit	Kleinste binäre Informationseinheit
Blob	Binary Large Object, Grosse binäre Datenobjekte
Byte	Masseinheit in der Technik und besteht aus einer Folge von 8 Bits
C	
Capture	Erfassung von Netzwerkdaten mit Wireshark
Certificate	Durch kryptografische Verfahren werden die Authentizität und Integrität geprüft
Channel	Kanal im Netzwerk
Chiffretext	Verschlüsselte Texte
Code Execution	Ausführung von Code
Convert	Umwandlung
Credentials	Anmeldeinformationen vom Benutzer
CredSSP	Protokoll um Anmeldeinformationen sicher zu übermitteln
D	
Denial of Service	Server ausser Betrieb setzen oder unzugänglich machen durch sehr viele Anfragen
DES	Data Encryption Standard, Symmetrischer Verschlüsselungsalgorithmus
Diffi-Hellman	Asymmetrisches kryptographisches Verfahren für Schlüsselaustausch

Docker	Docker ist eine Container Virtualisierung der isolierten freien Software.
DoD	Abkürzung für (Definition of Done)
Domain	Netzwerkumgebung
Domänencontroller (DC)	Server für zentrale Authentifizierung von Computern
E	
Enum	Datentyp für Aufzählungstypen, welche Konstanten bereitstellt
Eventdispatcher	Leitet Events an zuständige Funktionen weiter
F	
Flag	Eine Variable, die einen bestimmten Zustand angibt
G	
GIF	Graphics Interchange Format, Grafikformat
Git	Tool für Versionskontrolle.
GitLab	Entwicklungsplattform.
GSS-API	Schnittstelle für Anwendung, welche auf Sicherheitsdienste zugreift
GUI	Graphical User Interface, Grafische Oberfläche von einem Programm
H	
Handshake	Verbindungsaufbau mit gegenseitiger Bestätigung
Hash	Kryptografischer Funktion mit Output von fester Länge
HMAC	Authentifizierungscode von Nachrichten
Host	Rechner im Netzwerk
HTTP(S)	Hypertext Transfer Protocol (Secure), Kommunikationsprotokoll im World Wide Web
I	
Image (Docker)	Es enthält Code, Library und Tools, um eine Applikation zu starten.
ISO	Eine Archivdatei wie eine CD oder DVD.

K

Kerberos	Verteilter Authentifizierungsdienst.
Key Distribution Center	Schlüssel Verwaltungszentrale in Kerberos.

L

Layer	Schichten im Netzwerk
Linux	UNIX Betriebssystem

M

macOS	Betriebssystem von Apple
Malware	Software, die in Computer eindringen und Schäden verursachen
Mapping	Verknüpfung von Feldern
Master Secret	Schlüssel, die aus Premaster Secret abgeleitet werden für TLS Verbindung
MCS	Multipoint Communication Service Protocol, Teil des RDP Protokolls
MD4	Message Digest 4, kryptografische Hashfunktion
MITM	Man in the Middle, Angriffsart, bei der sich Hacker zwischen zwei Teilnehmern dazwischenschaltet

N

Netzwerkframework	Ein Programmiergerüst für Netzwerk.
NLA	Network Level Authentication
Nonce	Eine zufällige Zeichenfolge, welche einmalig verwendet wird.
NTLM	NT LAN-Manager, Authentifizierungsdienst.

O

Observer	Design-Pattern aus dem Bereich der Softwareentwicklung.
OEM-String	Freiform-Informationen, die im System-Management-BIOS gespeichert sind.
OpenSSL	Kostenlose Software für Transport Layer Security.
OS X	Betriebssystem von Apple.

P

Package-Diagramm	Zeigt eine bestimmte Sicht auf die Struktur des modellierten Systems.
Parser	Programm, das in der Informatik für die Zerlegung und Umwandlung von Daten zuständig ist.
Pass-Through-Authentifizierung	Dienst, der auf einem oder mehreren lokalen, domänenverbundenen Servern läuft und die Anmeldung eines Benutzers validiert.
Payload	Zwischen Kommunikationsteilnehmer transportierten Daten eines Datenpakets, die keine Steuer- oder Protokollinformationen enthalten.
PDU	Protocol Data Unit, Kommunikationsprotokoll, die der Kommunikation zwischen gleichberechtigten Protokollschichten dienen.
Pentest	Penetrationstest, Umfassender Sicherheitstest einzelner Computer oder Netzwerke.
Perfect Forward Secrecy	Eigenschaft bestimmter Schlüsselaustauschprotokolle mit dem Ziel, einen gemeinsamen Sitzungsschlüssel zu vereinbaren, so dass diese nicht rekonstruiert werden kann.
PFX	Dateiendung für ein Zertifikat in dem Format PKCS#12.
PoC (Proof of Concept)	Nachweis der prinzipiellen Machbarkeit eines Projekts.
PowerPC	Mikroprozessor-Architektur aus 1991 von Apple, IBM und Motorola.
Premaster Secret	Wert, der direkt aus dem Schlüsselaustausch stammt.
Private Key	Geheimer Entschlüsselungsschlüssel im asymmetrischen Krypto System.
Public Key	Öffentlicher Verschlüsselungsschlüssel im asymmetrischen Krypto System.
Pyrdp	RDP Man-in-the-Middle Tool
Python	Universelle, höhere Programmiersprache.

R

Rainbow-Table	Datenstruktur, die eine schnelle, speichereffiziente Suche nach der ursprünglichen Zeichenfolge für einen gegebenen Hashwert ermöglicht.
Random	Zufällig generierter Wert.
Random Key	Verschlüsselungsschlüssel, der zufällig generiert wurde.
RC4	Ron's Code 4, Verschlüsselungsverfahren für Streams.

RDP	Remote Desktop Protokoll, Netzwerkprotokoll, das den Fernzugriff auf andere Rechner ermöglicht.
RSA	Rivest Shamir Adleman, Asymmetrisches kryptographisches Verfahren zum Verschlüsseln und Signieren.
RUP	Rational Unified Process, Vorgehensmodell für die Durchführung von Softwareprojekten.
S	
Session	Stehende Verbindung eines Clients mit einem Server.
SHA	Secure Hash Algorithm, Eine Gruppe standardisierter kryptologischer Hashfunktionen.
Signatur	Verschlüsselte Daten, um die Integrität des Senders zu verifizieren.
SP (Service Pack)	Windows-Update, in dem bereits veröffentlichte Updates kombiniert werden.
SPNEGO	GSSAPI-Verhandlungsmechanismus für die Authentifizierung.
SSL	Secure Socket Layer, Vorgänger von TLS, Verschlüsselungsprotokoll zur sicheren Datenübertragung im Internet.
SSPI	Schnittstelle und Komponente der Windows-API für Security Service Provider.
Symmetrischer Schlüssel	Beide Kommunikationspartner verwenden den gleichen Schlüssel für die Verschlüsselung und Entschlüsselung.
T	
T.125	Kommunikationsprotokoll
Threading	Handhabung von leichtgewichtigen Prozessen.
Ticket Granting-Server	Key Distribution Center (KDC), das vom Kerberos-Protokoll als vertrauenswürdige dritte Partei verwendet wird.
TLS	Transport Layer Security, Verschlüsselungsprotokoll zur sicheren Datenübertragung im Internet.
Token	Hardware-/Software-Komponente zur Identifizierung und Authentifizierung von Benutzern.
TPKT	Wird vom TCP-Protokoll verwendet, um mehrere Nachrichten in einem Paket zu senden.
TSRequest	Vom CredSSP-Client und CredSSP-Server verwendete Nachrichtenstruktur.
U	
Ubuntu	Linux-Betriebssystem, das auf Debian basiert.
Unicode	Internationaler Standard für Schriftzeichen.

UNIX	Betriebssystem
V	
VirtualBox	Virtualisierungssoftware des US-amerikanischen Unternehmens Oracle.
W	
Windows	Betriebssystem der Unternehmen Microsoft.
Wireshark	Software zur Analyse und grafischen Aufbereitung von Datenprotokollen.
X	
X.509	ITU-T-Standard für eine Public-Key-Infrastruktur zum Erstellen digitaler Zertifikate.
X.224	Transport-Protokoll

13 Anhang B: Literaturverzeichnis

Generic Security Service Application Program Interface Version 2, Update 1. (01 2000). Abgerufen am 13. 05 2021 von https://www.rfc-editor.org/rfc/rfc2743.txt

Bordes, A., Ebalard, A., & Rigo, R. (15. 06 2021). *ANSSI*. Von Sécurité de RDP: https://www.ssi.gouv.fr/uploads/IMG/pdf/Securite_de_RDP_article.pdf abgerufen

Bourguenolle, T., & Bertoli, G. (15. 06 2021). *GitHub*. Von CredSSPY: https://github.com/croustibaie/CredSSPY abgerufen

Dierks, T., & Rescorla, E. (08 2008). *IETF Documents*. Von The Transport Layer Security (TLS) Protocol Version 1.2: https://tools.ietf.org/html/rfc5246 abgerufen

FreeRDP. (15. 06 2021). Von https://raw.githubusercontent.com/wiki/FreeRDP/FreeRDP/files/pcap/nla_win7_win2k8r2.zip abgerufen

GitHub. (15. 06 2021). Von Seth: https://github.com/SySS-Research/Seth abgerufen

GitHub. (15. 06 2021). Von rdpy: https://github.com/citronneur/rdpy abgerufen

Glass, E. (2006). *The NTLM Authentication Protocol and Security Support Provider*. Abgerufen am 13. 05 2021 von http://davenport.sourceforge.net/ntlm.html

GoSecure. (17. 11 2020). *GitHub*. Von pyrdp: https://github.com/GoSecure/pyrdp abgerufen

IONOS. (13. 02 2019). Von Man-in-the-Middle-Attack: Angriffsmuster und Gegenmaßnahmen: https://www.ionos.de/digitalguide/server/sicherheit/man-in-the-middle-attack-angriffsmuster-im-ueberblick/ abgerufen

IONOS. (05. 06 2020). Von NTLM: Wie funktioniert das Protokoll?: https://www.ionos.de/digitalguide/server/knowhow/ntlm-nt-lan-manager/ abgerufen

Kryptowissen.de. (01. 09 2016). Von Kerberos: https://www.kryptowissen.de/kerberos.php abgerufen

Linn, J. (01 2000). *IETF Documents*. Von Generic Security Service Application Program Interface: https-//www.rfc-editor.org/rfc/rfc2743.txt.webloc abgerufen

Microsoft. (31. 05 2018). Von SSPI Architectural Overview: https://docs.microsoft.com/en-us/windows/win32/rpc/sspi-architectural-overview abgerufen

Microsoft. (07. 04 2021). Von RDP Connection Sequence: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-rdpbcgr/023f1e69-cfe8-4ee6-9ee0-7e759fb4e4ee#:~:text=The%20goal%20of%20the%20RDP,processed%20between%20client%20and%20server. abgerufen

Microsoft. (07. 04 2021). Von CredSSP Overview: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-cssp/85f57821-40bb-46aa-bfcb-ba9590b8fc30 abgerufen

Microsoft. (15. 06 2021). Von NT LAN Manager (NTLM) Authentication Protocol: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-nlmp/b38c36ed-2804-4868-a9ff-8dd3182128e4 abgerufen

MS-SPNG. (07. 04 2021). Abgerufen am 13. 05 2021 von https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-spng/f377a379-c24f-4a0f-a3eb-0d835389e28a

Petters, J. (10. 08 2020). *VARONIS*. Von What is a Man-in-the-Middle Attack: Detection and Prevention Tips: https://www.varonis.com/blog/man-in-the-middle-attack/ abgerufen

PyPi. (15. 06 2021). Von Twisted 21.2.0: https://pypi.org/project/Twisted/ abgerufen

PyPi. (15. 06 2021). Von pyOpenSSL 20.0.1: https://pypi.org/project/pyOpenSSL/ abgerufen

PyPi. (15. 06 2021). Von impacket 0.9.23: https://pypi.org/project/impacket/ abgerufen

PyPi. (15. 06 2021). Von requests-credssp 1.2.0: https://pypi.org/project/requests-credssp/ abgerufen

PyPi. (16. 06 2021). Von tk-tools 0.14.0: https://pypi.org/project/tk-tools/ abgerufen

pyrdp. (13. 04 2021). Abgerufen am 13. 05 2021 von https://github.com/GoSecure/pyrdp

Reiner, Shaked. (04. 07 2020). *cyberark*. Von Explain Like I'm 5: Remote Desktop Protocol (RDP): https://www.cyberark.com/resources/threat-research-blog/explain-like-i-m-5-remote-desktop-protocol-rdp abgerufen

Suau, R. (23. 05 2019). *pradeo*. Von https://blog.pradeo.com/de/was-ist-man-in-the-middle-angriff abgerufen

The Transport Layer Security (TLS) Protocol Version 1.2. (08 2008). Abgerufen am 13. 05 2021 von https://datatracker.ietf.org/doc/html/rfc5246

Wikipedia. (11. 11 2020). Von SPNEGO: https://en.wikipedia.org/wiki/SPNEGO abgerufen

Wikipedia. (06. 02 2021). Von Security Support Provider Interface: https://en.wikipedia.org/wiki/Security_Support_Provider_Interface abgerufen

Wikipedia. (24. 02 2021). Von Remote Desktop Protocol: https://de.wikipedia.org/wiki/Remote_Desktop_Protocol abgerufen

Wikipedia. (31. 03 2021). Von Network Level Authentication: https://en.wikipedia.org/wiki/Network_Level_Authentication abgerufen

Wikipedia. (19. 04 2021). Von Generic Security Services Application Program Interface: https://en.wikipedia.org/wiki/Generic_Security_Services_Application_Program_Interface abgerufen

Wikipedia. (02. 04 2021). Von NT LAN Manager: https://en.wikipedia.org/wiki/NT_LAN_Manager abgerufen

Wikipedia. (15. 06 2021). Von Kerberos (protocol): https://en.wikipedia.org/wiki/Kerberos_(protocol) abgerufen

wolfSSL. (15. 06 2021). Von TLS 1.3 Performance Analysis – Full Handshake: https://www.wolfssl.com/tls-1-3-performance-part-2-full-handshake-2/ abgerufen

Zhu, L., Leach, P., Jaganathan, K., & Ingersoll, W. (10 2006). *IETF Documents*. Von The Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism: https://tools.ietf.org/html/rfc4178 abgerufen

14 Anhang C: Abbildungsverzeichnis

15 Anhang D: Tabellenverzeichnis

16 Anhang E: Sequenzdiagramm mit MITM

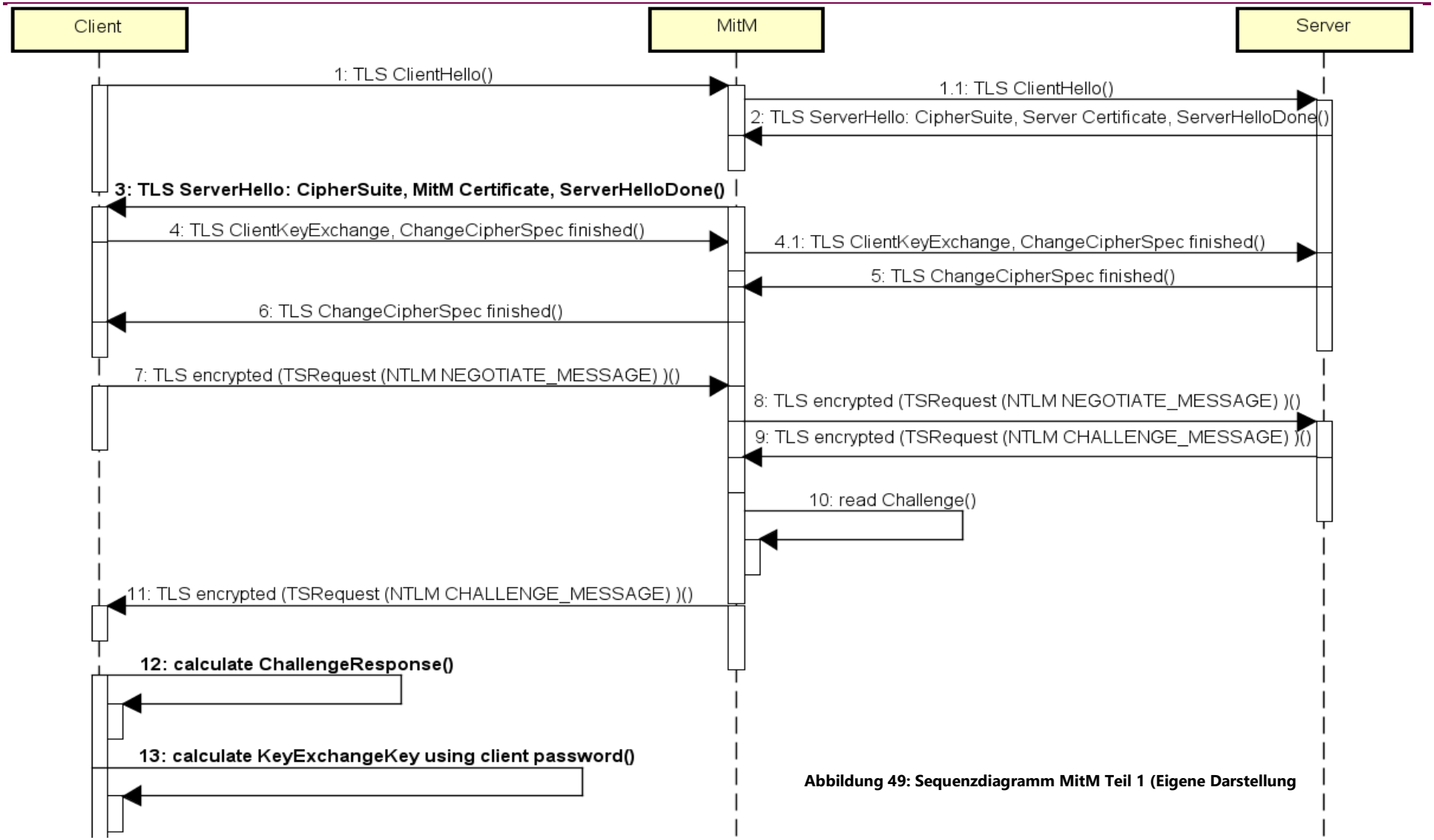


Abbildung 49: Sequenzdiagramm MitM Teil 1 (Eigene Darstellung

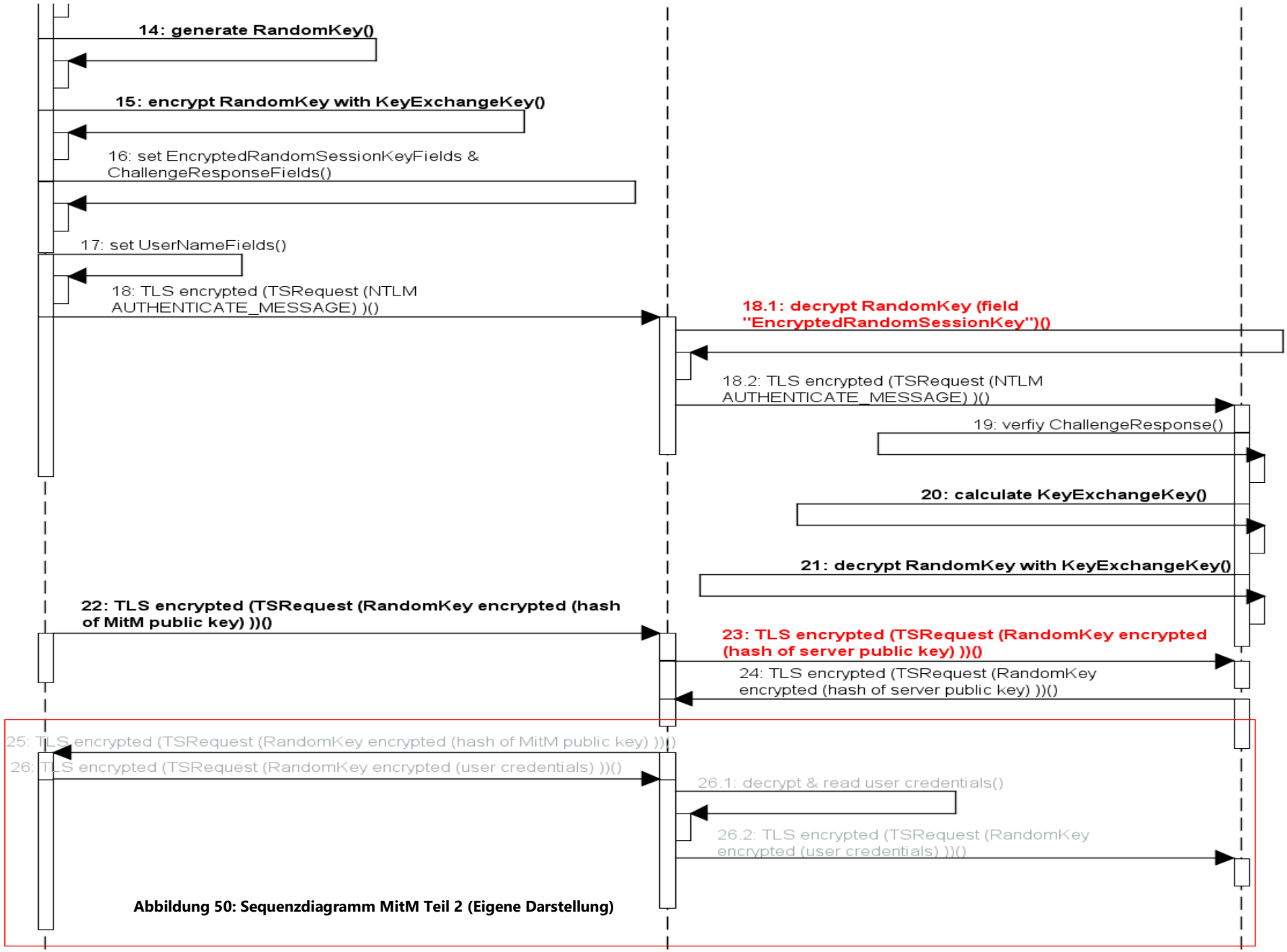


Abbildung 50: Sequenzdiagramm MitM Teil 2 (Eigene Darstellung)

17 Anhang F: Fake-Login-Screen Code

```
# coding=utf-8


from tkinter import *
from tkinter import messagebox
from tkinter.ttk import *
from tkinter import filedialog
from tkinter import Menu
from tkinter import Label
from tkinter import Entry
from tkinter import Button
from PIL import ImageTk, Image


window = Tk()
window.attributes('-fullscreen', True)
window.title("Login")
window.geometry('1550x850')
window.resizable(False, False)


class BackGround(Frame):
    def __init__(self, master, *pargs):
        Frame.__init__(self, master, *pargs)
        self.image = Image.open("images/WindowsLockScreen7.png")
        self.img_copy= self.image.copy()
        self.background_image = ImageTk.PhotoImage(self.image)
```

```
        self.background = Label(self, image=self.background_image)
        self.background.pack(fill=BOTH, expand=YES)
        self.background.bind('<Configure>', self._resize_image)

    def _resize_image(self, event):
        new_width = event.width
        new_height = event.height
        self.image = self.img_copy.resize((new_width, new_height))
        self.background_image = ImageTk.PhotoImage(self.image)
        self.background.configure(image =  self.background_image)

e = BackGround(window)
e.pack(fill=BOTH, expand=YES)

txt_username = Entry(window, font=("Segoe UI", 13), bd=2, bg="white", insertofftime=600, insertwidth
="1p", highlightthickness=1)
txt_username.config(highlightbackground = "gray", highlightcolor= "#eaeaea")
txt_username.place(relx=0.5, rely=0.61, anchor=CENTER, height=40, width=290)
txt_username.focus()

txt_password = Entry(window, show="•", font=("Segoe UI", 20), bd=2, bg="white", insertofftime=600, i
nsertwidth="1p", highlightthickness=1)
txt_password.config(highlightbackground = "gray", highlightcolor= "gray")
txt_password.place(relx=0.489, rely=0.675, anchor=CENTER, height=40, width=259)
txt_password.configure(cursor="xterm")

frameCnt = 50
frames = [PhotoImage(file='images/WindowsLoadingScreenSmall.gif',format = 'gif -
index %i' %(i)) for i in range(frameCnt)]
```

```
def update(ind):
    frame = frames[ind]
    ind += 1
    if ind == frameCnt:
        ind = 0
    label.configure(image=frame)
    window.after(100, update, ind)


def clicked(event=None):
    print("Username: " + txt_username.get() + "\nPassword: " + txt_password.get())
    can.place(relx=-0.1, rely=-0.1)
    label.place(relx=0.42, rely=0.35)
    window.after(1, update, 0)


login_image = PhotoImage(file = "images/LoginButton.png")
btn_login = Button(window, text = "Login", image=login_image, command=clicked, width=33, height=
33, highlightthickness=1)
btn_login.config(highlightbackground = "gray", highlightcolor= "gray")
btn_login.place(relx=0.57, rely=0.653)


window.bind('<Return>', clicked)


can = Canvas(window, width=20000, height=20000, bg="#044a91", borderwidth=0)
label = Label(window, borderwidth=0)
window.mainloop()
```

18 Anhang G: CredSSP-Authentifizierung Code

```
def clientConnect(host, username, password, connect = True):
    tpkt = TPKT()
    tpdu = TPDU()
    rdp_neg = RDP_NEG_REQ()
    rdp_neg['Type'] = TYPE_RDP_NEG_REQ
    rdp_neg['requestedProtocols'] = PROTOCOL_HYBRID_EX | PROTOCOL_HYBRID | PROTOCOL_SSL
    tpdu['VariablePart'] = rdp_neg.getData()
    tpdu['Code'] = TDPU_CONNECTION_REQUEST
    tpkt['TPDU'] = tpdu.getData()
    s = socket.socket()
    s.connect((host,3389))
    s.sendall(tpkt.getData())
    pkt = s.recv(8192)
    tpkt.fromString(pkt)
    tpdu.fromString(tpkt['TPDU'])
    cr_tpdu = CR_TPDU(tpdu['VariablePart'])
    if cr_tpdu['Type'] == TYPE_RDP_NEG_FAILURE:
        rdp_failure = RDP_NEG_FAILURE(tpdu['VariablePart'])
        rdp_failure.dump()
        logging.error("Server doesn't support PROTOCOL_HYBRID, hence we can't use CredSSP to check credentials")
        return
    else:
          rdp_neg.fromString(tpdu['VariablePart'])
```

Schritt 1: Starten der SSL-Verbindung

```
ctx = SSL.Context(SSL.TLSv1_2_METHOD)

ctx.set_cipher_list('RC4,AES')

tls = SSL.Connection(ctx,s)

tls.set_connect_state()

tls.do_handshake()
```

Schritt 2: Senden des SPNEGO-Handshakes

```
auth = ntlm.getNTLMSSPType1('','',True, use_ntlmv2 = True)


ts_request = TSRequest()

ts_request['NegoData'] = auth.getData()


tls.send(ts_request.getData())

buff = tls.recv(4096)

ts_request.fromString(buff)
```

Schritt 3: Beantwortung der Challenge und Signieren des Server-Public Keys

```
lmhash =''

nthash=''

type3, exportedSessionKey = ntlm.getNTLMSSPType3(auth, ts_request['NegoData'], username, password, lmhash, nthash, use_ntlmv2 = True)


server_cert =  tls.get_peer_certificate()

pkey = server_cert.get_pubkey()

dump = crypto.dump_privatekey(crypto.FILETYPE_ASN1, pkey)


dump = dump[7:]
```

```
pubKeyStr = b'\x30'+ asn1encode(dump)
clientNonce = "A"*32
magic = b"CredSSP Client-To-Server Binding Hash\x00"
h2 = hashlib.sha256()
h2.update(magic)
h2.update(clientNonce)
h2.update(pubKeyStr)

cipher = SPNEGOCipher(type3['flags'], exportedSessionKey)
signature, cripted_key = cipher.clientEncrypt(h2.digest())
ts_request['NegoData'] = type3.getData()
ts_request['clientNonce']=clientNonce
ts_request['pubKeyAuth'] = signature.getData() + cripted_key
try:

    tls.send(ts_request.getData())
    buff = tls.recv(1024)
except Exception as err:
    if str(err).find("denied") > 0:
        logging.error("Access Denied")
    else:
        print(err)
    return
```

Schritt 4: Überprüfung der Public Key-Signatur des Servers

```
try :
    ts_request = TSRequest(buff)


    signature, plain_text = cipher.decrypt(ts_request['pubKeyAuth'])
    if username != ""  and password != "":
       print "Found valid credentials: " +username+":"+password
except Exception as err:
    return 0
```

Schritt 5: Senden der verschlüsselten Anmeldeinformationen an den Server

```
if connect == True :
    tsp = TSPasswordCreds()
    tsp['userName']   = username
    tsp['password']   = password
    tsc = TSCredentials()
    tsc['credType'] = 1
    tsc['credentials'] = tsp.getData()
    signature, cripted_creds = cipher.clientEncrypt(tsc.getData())
    ts_request = TSRequest()
    ts_request['authInfo'] = signature.getData() + cripted_creds
    tls.send(ts_request.getData())
    print "Credentials sent"
    return tls
else :
    return 1
```

19 Anhang H: Risikotabelle

Nr	Titel	Beschreibung	max. Schaden [h]	Eintrittswahrscheinlichkeit	Gewichteter Schaden	Vorbeugung	Verhalten beim Eintreten
R1	Kommunikation	Ungenügende Kommunikation zwischen den Teammitgliedern und/oder dem Betreuer.	63	10%	6.3	Wöchentliche Team- und Review-Meetings durchführen.	Zusätzliche Kommunikation (Mail, Meeting, etc.) planen und durchführen.
R2	Komplexität	Komplexität der Arbeit ist zu hoch.	42	80%	33.6	Weiteres Vorgehen in den wöchentlichen Review-Meetings mit Betreuer besprechen und planen.	Schwierigkeiten und Unklarheiten mit Betreuer besprechen.
R3	Lernkurve	Fehlendes Vorwissen zum Thema oder unzureichende Kenntnisse in Umgang mit Tools.	126	45%	56.7	Wissen und Kenntnisse in der Elaboration-Phase aufbauen.	Teammitglieder sind gezwungen das nötige Wissen anzueignen. Bei grösseren Unklarheiten Betreuer einbeziehen.
R4	Ressourcenmangel	Es stehen keine ausreichenden Ressourcen (Hardware & Software) zur Verfügung.	21	5%	1.05	Frühzeitig die benötigen Ressourcen beantragen.	Nach alternativen Ressourcen oder Ausführmöglichkeiten umsehen.

R5	Ausfall externer Komponente	Ausfall von externen Komponenten (GitLab, Dropbox, Virtueller Server, Netzwerk)	21	5%	1.05	Alle Daten zusätzlich lokal sichern (inkl. inkrementelle Datensicherung).	Daten aus dem Backup wiederherstellen. Lokale virtuelle Maschinen aufsetzen und verwenden.
R6	Umsetzbarkeit der Aufgabenstellung	Die Aufgabenstellung ist nicht umsetzbar.	126	25%	31.5	Umsetzbarkeit der Aufgabe wird in der Elaboration-Phase genau untersuchen.	Mit Betreuer absprechen.
R7	Ungenaue Aufwandschätzung	Ungenaue Aufwandschätzungen führen zu Problem bezüglich Termine.	20	35%	7	Alle offenen Issues wöchentlich überprüfen und neu schätzen.	Zeitliche Planung des Projektes anpassen.
R8	Installation & Integration	Verzögerungen/Schwierigkeiten beim Aufbau von Hard- und Software. Erschwerte/Unmögliche Integration in das existierende Tool.	42	50%	21	Installationen anhand Anleitung durchführen. Integrationen gemäss Best-Practices durchführen.	Für Hilfestellung an Betreuer wenden.
R9	Ausfall Personal	Teammitglied oder Betreuer ist aus gesundheitlichen oder anderen Gründen abwesend.	21	15%	3.15	Falls Abwesenheit frühzeitig bekannt ist, weiter Vorgehen vorplanen. Genügend Reserve-Zeit einplanen.	Reserve-Zeit nutzen und zeitliche Planung der Aufgaben neu erstellen.
Summe			**461**		**158.2**		

Tabelle 12: Risikotabelle

20 Anhang I: Fehlermeldung CredSSP Authentifizierung Implementierung

```
/home/user/Desktop/rdp-man-in-the-middle/code/pyrdp/venv/bin/python /home/user/Desktop/rdp-man-in-the-middle/code/pyrdp/bin/pyrdp_mitm.py 152.96.56.45 --username Administrator --p
[2021-06-11 18:07:46,782] - INFO - GLOBAL - pyrdp.mitm - Target: 152.96.56.45:3389
[2021-06-11 18:07:46,782] - INFO - GLOBAL - pyrdp.mitm - Output directory: /home/user/Desktop/rdp-man-in-the-middle/code/pyrdp/bin/pyrdp_output
[2021-06-11 18:07:46,783] - INFO - GLOBAL - pyrdp - MITM Server listening on 0.0.0.0:10009
152.96.56.45
first reactor <twisted.internet.asyncioreactor.AsyncioSelectorReactor object at 0x7f8bb432ab80>
[2021-06-11 18:07:52,952] - INFO - Amanda342059 - pyrdp.mitm.connections.tcp - New client connected from 127.0.0.1:48042
[2021-06-11 18:07:52,952] - INFO - Amanda342059 - pyrdp.mitm.connections.x224 - No cookie for this connection
[2021-06-11 18:07:52,957] - INFO - Amanda342059 - pyrdp.mitm.connections.tcp - Server connected
[2021-06-11 18:07:52,957] - INFO - Amanda342059 - pyrdp.mitm.connections - NLA is activated on Server
startTLS
startNegoData
ntlm auth
TSRequestPDU{'payload': b'0/\xa0\x03\x02\x01\x06\xa1(0&0$\xa0"\x04 NTLMSSP\x00\x01\x00\x00\x005\x82\x88\xe0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'}
sent
[2021-06-11 18:07:53,002] - INFO - Amanda342059 - pyrdp.mitm.connections.tcp - Server connection closed. [('SSL routines', 'ssl3_read_bytes', 'tlsv1 alert internal error')]
```

Abbildung 51: Fehlermeldung CredSSP Authentifizierung Implementierung (Eigene Darstellung)

21 Anhang J: Persönliche Reflexion

21.1 **Kevin Moro**

Für meine Kollegen und mich war das Thema RDP Man-in-the-Middle unbekanntes Terrain, da wir bisher wenig im Bereich Informationssicherheit gemacht haben. Trotzdem war dieses Thema für uns bei der Auswahl des Bachelorarbeitsthema der Favorit. Wir haben uns auch entsprechend gefreut, als wir dann dieses Thema bekommen haben.

Der Start in die Bachelorarbeit war etwas schwierig, da wir noch keine Erfahrungen gemacht haben mit diesem Typ von Arbeit und auch keine Erfahrungen in diesem Fachgebiet hatten. Unser Betreuer hat uns dann geholfen, ins Thema einzusteigen und hat uns gut unterstützt. Das hat sehr geholfen, um uns in das Thema einzuarbeiten und auch um das Vorgehen zu planen.

Die Herangehensweise an dieses Projekt unterscheidet sich stark von Projekten, die wir bisher gemacht haben. Wir mussten uns am Anfang sehr stark in das Thema einlesen, da wir einerseits noch kein Wissen über die Thematik hatten und andererseits ist dieses Thema auch sehr umfangreich. Diese Analyse hat einen grossen Teil unserer Arbeit eingenommen und das war für uns neu, da wir sonst die meiste Zeit in einem Projekt mit der Implementierung verbracht haben.

Das war aber eine interessante Erfahrung, ein Projekt zu machen, indem es nicht nur um die Entwicklung geht, sondern vor allem um die Forschung/Analyse, aber auch das Element des Entwickelns beinhaltete.

Als wir dann durch die Analyse festgestellt haben, dass der Man-in-the-Middle, so wie in der Aufgabenstellung beschrieben, nicht funktioniert, hatten wir für eine kurze Zeit eine Ungewissheit, wie es mit der Arbeit weitergehen soll. Jedoch konnten wir schnell einen anderen Ansatz für einen Man-in-the-Middle Angriff finden und nach Absprache mit unserem Betreuer konnten wir mit diesem Ansatz weiterfahren.

Die Arbeit mit Danusan Premananthan und Aynkaran Sundralingam war für mich stehts positiv und wir haben uns immer gut verstanden. Unklarheiten konnten schnell beseitigt werden mit Meetings über Microsoft Teams oder vor Ort am Campus in Rapperswil.

Diese Bachelorarbeit war für mich eine interessante Erfahrung und ich konnte dadurch mein Wissen in diesem Bereich sehr vertiefen.

Ich bin mit unserer Bachelorarbeit zufrieden, auch wenn wir einen Teil der Implementation leider nicht fertigstellen konnten. Sonst haben wir jedoch alle Aufgaben erfüllt.

21.2 **Danusan Premananthan**

Das Thema RDP Man-in-the-Middle war für meine Kollegen und mich Neuland, jedoch waren wir neugierig und bereit, diese Herausforderung anzugehen. Trotz einiger Schwierigkeiten und mangelnder Erfahrung mit diesem Thema, denke ich, dass die Bachelorarbeit gut gelaufen ist. Da wir uns gut kannten und bereits andere grosse Aufgaben und Projekte gemeinsam durchgeführt hatten, kannten wir die Schwächen und Stärken des anderen und konnten die Aufgaben entsprechend planen und verteilen.

Die Arbeit zwingt uns zu einer anderen Herangehensweise als wir es bei einem Projekt, bei dem es nur um die Entwicklung einer Applikation geht, gewohnt waren. In den letzten Projekten konnten wir immer früh eine Spezifikation und ein Konzept entwerfen. Diesmal brauchte es erst eine Analyse und Wissensbeschaffung, bevor wir mit der Umsetzung starten konnten.

Bei der Implementierung des Proof of Concepts in PyRDP wurde zu Beginn grob in drei Aufgaben unterteilt. Allerdings war die Implementierung im PyRDP-Projekt komplizierter als ursprünglich gedacht und konnte nur zwei der Implementierungsaufgaben abschliessen. Im Nachhinein wäre eine frühere Analyse von PyRDP Code hilfreich gewesen.

Die Zusammenarbeit mit Aynkaran Sundralingam und Kevin Moro war für mich immer positiv und es gab keine Interessenkonflikte. Zudem konnten Probleme und Anregungen schnell über Microsoft Teams, Telefon oder am Campus in Rapperswil geklärt werden.

Im Ganzen bin ich mit der Bachelorarbeit zufrieden. Am Ende konnten wir bis auf die Implementierung des Layer Mappings alle Ziele der Aufgabenstellung erreichen und dabei das Wissen in verschiedenen Gebieten erweitern.

21.3 **Aynkaran Sundralingam**

Da ich mich persönlich sehr für das Thema Informationssicherheit interessiere, war «RDP Man-in-the-Middle» für mich von Anfang an ein Favorit bei der Auswahl der Themen für die Bachelorarbeit. Da auch meine Kollegen, Kevin Moro und Danusan Premananthan, mit denen ich schon mehrere Projekte erfolgreich durchgeführt habe, in diesem Semester ebenfalls im Abschlusssemester sind und auch die Bachelorarbeit schreiben müssen, war es für mich klar, dass auch diese abschliessende Arbeit gemeinsam absolviert wird. Die Zusammenarbeit und die Durchführung der Bachelorarbeit hat auch diesmal gut funktioniert, da wir die Arbeitsweise und die Stärken sowie Schwächen des jeweils anderen bereits aus früheren Projekten kannten.

Am Anfang der Arbeit standen wir jedoch vor einer grossen Herausforderung, da keiner von uns wirklich Erfahrung in der Durchführung einer Machbarkeitsanalyse zu einem Informationssicherheitsthema und generell in diesem Fachgebiet hatte. Daher war die Vorgehensweise bei dieser Arbeit zunächst unklar. Unser Betreuer, Cyrill Brunschwiler, hat uns geholfen, in das Thema einzusteigen und hat uns ausreichende Unterstützung geboten. Dies ermöglichte es uns, schnell in das «fremde» Thema einzusteigen und das notwendige Vorgehen für diese Arbeit zu planen.

Es war eine interessante Erfahrung, ein Projekt in Angriff zu nehmen, dass sich nicht nur auf die Entwicklung einer Applikation beschränkte, sondern auch ein hohes Mass an Analyse von uns verlangte. Da wir durch die Machbarkeitsanalyse feststellen mussten, dass unser Vorhaben gemäss der Aufgabenstellung nicht umsetzbar war, herrschte für kurze Zeit Ungewissheit, wie die Arbeit weitergeführt werden sollte. Zu diesem Zeitpunkt hatten sie jedoch bereits genug Wissen über das Thema erworben, um einen alternativen Ansatz zur Durchführung des Angriffes zu entwerfen. Obwohl wir die Implementierung eines Man-in-the-Middle (RDP Enhanced Security NLA) Proof of Concept für PyRDP aus Zeitgründen nicht vollständig abschliessen konnten, bin ich zufrieden mit dem, was wir erreicht haben.

Diese Bachelorarbeit hat mir die Möglichkeit gegeben, mein Wissen in diesem Themenbereich zu erweitern und erste Erfahrungen in der Herangehensweise an solche Aufgaben zu sammeln. Durch diese Arbeit ist mein Interesse an diesem Themenbereich auf jeden Fall gestiegen.