目录

概论

本文简要介绍 RDP 协议和对 rdesktop1.7.0 程序结构作一些分析。在 RDP 协议一节里，将对 RDP4 和 RDP5 作一些介绍，但在 rdesktop 程序结构一节里并不区分 RDP4 和 RDP5 版本。因为网上关于 rdesktop 的资源非常少，对于 rdesktop 笔者也是在探索阶段，看了半个月的代码，对 rdesktop 的软件结构也有了一定的了解。要完全读懂 rdesktop 所有代码，目前，对于笔者来说并不现实。本文仅对 rdesktop 的 1.7.0 版本的源码结构进行分析，不涉及任何具体实现的问题。最后对 rdesktop 中最重要的 STREAM 结构作简单介绍。

1. RDP 网络协议

rdesktop 软件是基于 RDP 协议的，RDP 是微软制定的一套远程桌面协议标准，但因为它不是一个开放的协议，rdesktop 的开发人员采用逆向工程来研究 RDP 协议。在 windows 2008 系统中我们已经看到，微软的 RDP 协议已经有了 7.1 版本，但 rdesktop 目前只能支持 RDP 5 版本的协议。在介绍 rdesktop 源码结构之前，先介绍一下 RDP 的网络协议栈，这是 rdesktop 实现的基础。

和大多数的网络协议一样，RDP 协议由多个层次组成。图 1.1 是 RDP4 和 RDP5 协议栈的层次结构图。

1.1. RDP4 协议

协议栈的最底层 TCP 连接层，主要是从客户端连接到服务器端的 3389 端口。RDP 协议建立在 TCP/IP 协议之上，客户端与服务器端是通过网络进行数据传输，此层主要是建立网络的连接。

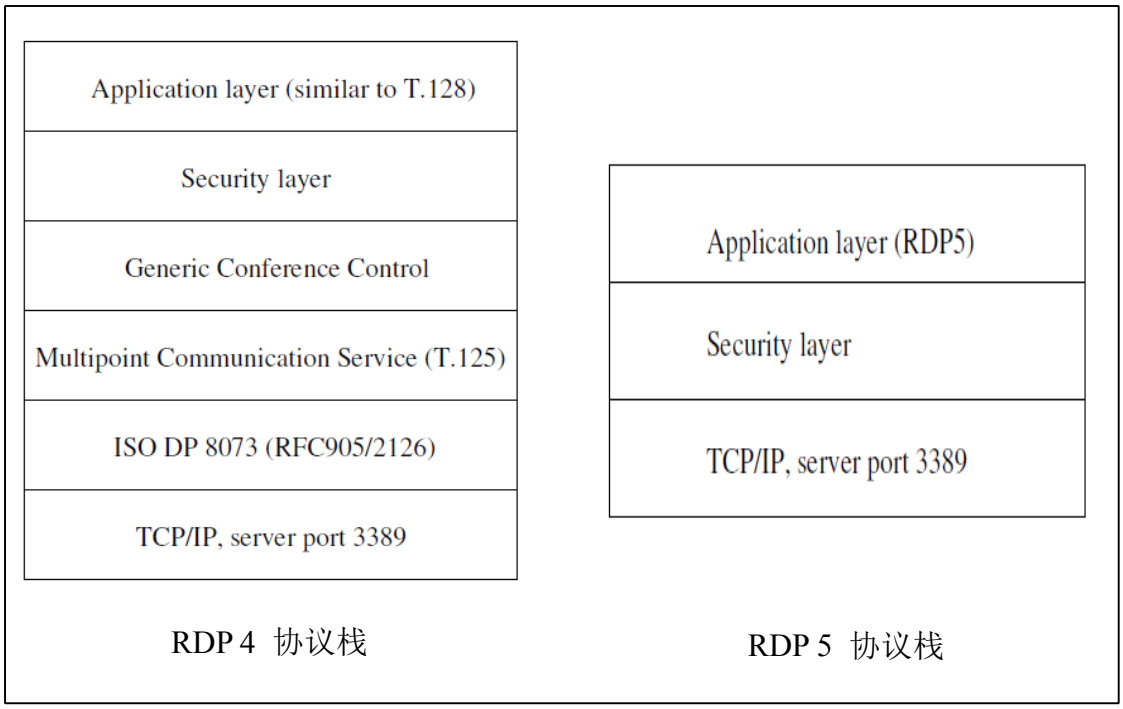

图 1.1 RDP 协议栈层次结构

在 TCP 层之上是 ISO 层，此层发送 ISO 数据包。它表示 RDP 数据的正常连接通信。并提供了一种能在普通 TCP 应用之上运行的标准。

在 ISO 层上是多点通信服务（Multipoint Communication Services）层，也称为 MCS 层，此层用于定义多个虚拟通道，用以拆分表示不同虚拟通道的数据。在 RDP5 里面，允许多个虚拟通道的应用，例如发送剪切板通道和声音数据通道等。

MCS 层之上的是通用会议控制（Generic Conference Control），此层在协议实现的时候几乎是不可见的，对用户来说是透明的。

安全层，也叫 SEC 层，它对 MCS 层传送的数据进行加解密。

协议栈的最顶层是 RDP 的应用层。应用层定义了图像数据如何传送到客户端，鼠标和键盘数据如何传送到服务器等。

1.2. RDP5 协议

从图 1.1 的对比中，明显看出，RDP4 和 RDP5 协议层次并不相同，微软对 RDP5 中作了重新定义，使得 RDP5 和 RDP4 之间有些差异。RDP5 的网络协议栈只有 3 层。RDP5 把 RDP4 协议中的 ISO、MCS 和 GCC 层取消，取而代之的是定义了一个更加紧凑的数据包格式。

值得注意的是，RDP5 协议可以看作是 RDP4 协议的一种改进，但并不说明 RDP5 协议中的数据包都是完完全全的 RDP5 类型，至少在建立连接阶段其数据格式就跟 RDP4 完全一样，所有从客户端到服务器的数据包就完全是 RDP4 的格式。RDP5 数据包格式只用于处理从服务器端传送过来的图像格式数据。其它通道（剪切板、声音等）仍然使用 RDP4 类型的数据包。

一个有趣的现象是使用旧协议（RDP4），理论上可以通过无 TCP/IP 连接运行 RDP，只需要重写服务器和客户端的一小部分代码就可以了。但用 RDP5 数据包已经无法实现这样的功能，因为很多抽象层已经地剥除。

2. rdesktop 结构分析

为了让 rdesktop 的维护工作变得更容易，这里对 rdesktop 的内部工作机制进行一个描述。

2.1. Rdesktop 源码文件介绍

本节对 rdesktop1.7.0 源码中各个文件的作用作简单的说明（部分内容是对外文资料的直接翻译，译文可能有误，仅供参考）。

rdesktop.c 本文件包含了 main 函数的入口，对命令行参数进行分析和处理，初始化窗口等，最后进入主程序的循环。

xwin.c 包含了连接到 X Server 图形接口的代码。

xproto.h X 窗口相关函数的声明。

rdp.c 包含了发送和接收 RDP 数据包的代码，大部分代码只在建立协议的时候会调用，rdp 层协议。

rdp5.c 包含处理 RDP5 数据包的代码，rdp 层协议。

mcs.c 包含分析和发送 MCS 数据包的代码，mcs 层协议。

iso.c 包含分析和发送 ISO 数据包的代码，iso 层协议。

tcp.c 包含发送和接收 TCP 数据包的代码，tcp 层协议。

ewmhints.c 包含窗口管理器的通信程序代码。

licence.c 包含许可证（license）处理函数。

orders.c 包含从服务器处理次序（图形数据）的代码。

parse.h 包含操作 STREAM 结构体的宏（同样在本文件定义）。这些宏主要用于从服务器读写数据，或者向服务器读写数据。

proto.h 声明各个源码文件中的公用函数原型。

types.h 定义了 rdesktop 中公共数据类型。

scancodes.h 包含了键盘相关的定义。

secure.c 包含了编解码相关的代码，例如分析公共密钥和对数据进行编码和解码。

xkeymap.c 用于 X 窗口系统的键盘映射。

bitmap.c 包含对来自服务器的位图（例如图标和其它小图像）进行解压的函数。

cache.c 包含位图、字体、桌面和指针等缓存函数。

channels.c 当 RDP5 通道上有数据时，注册的回调函数被调用。

cliprdr.c 主要是 X windows 系统和微软 Windows 之间，或者不同的 rdesktop 会话之间的剪切板数据处理。

constants.h 定义了 RDP 的各种常量。

disk.c/disk.h 磁盘重定向相关定义和函数。

lspci.c 支持 Matrox lspci 通道。

mppc.c RDP 解压相关函数。

printer.c 打印机相关的函数。

printfercache.c 打印机缓冲相关函数。

pstcache.c 持续的位图缓冲相关的函数。

rdpsnd.h/rdpsnd.c 音频相关定义和函数声明/音频通道处理函数。

rdpsnd_alsa.c 音频通道处理函数——ALSA 驱动相关函数。

rdpsnd_dsp.h/rdpsnd_dsp.c 音频 DSP 相关函数。

rdpsnd_libao.c 音频通道处理函数——libao 驱动相关函数。

rdpsnd_oss.c 音频通道处理函数——开放音频系统相关函数。

rdpsnd_sgi.c 音频通道处理函数——SGI/IRIX 相关函数。

rdpsnd_sun.c 声音通道处理函数——SUN 相关处理函数。

scard.h/scard.c 智能卡支持的相关定义和函数。

seamless.h/seamless.c 无缝支持 Windows 的相关函数。

serial.c 串口支持的相关函数。

ssl.h/ssl.c 安全套接字抽象层相关定义和函数。

xclip.c 剪切板相关函数。

2.2. rdesktop 函数调用层次

从图 1 中可以知道，RDP 协议的基本层次结构，从顶层到底层的结构分别为：RDP 数据层（rdp）、加密解密层（sec）、多点通信服务层（mcs）、ISO 数据层（iso）

和 TCP 网络连接层（tcp）。rdesktop 也按这样的层次来调用的，如图 2.1 所示。

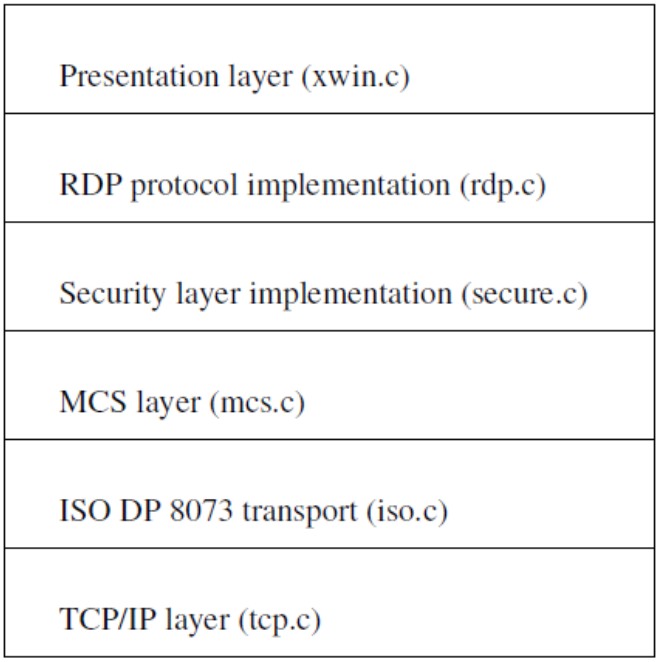

图 2.1 rdesktop 函数调用层次

在源码中很多地方都是按照 rdp_－>sec_－>mcs_－>iso_－>tcp_的层次进行调用，像 rdp 的连接函数、初始化函数、还有断开连接等操作都是按这样的层次进行调用。例如在 rdesktop 的源码中 rdp.c 文件中有 rdp_connect 函数。RDP 的连接过程如下所示：

rdp_connect－> sec_connect－> mcs_connect－> iso_connect－> tcp_connect

从上面的箭头方面可以知道，rdp_connect 要调用 sec_connect，而 sec_connect 要调用 mcs_connect 等，依此类推。RDP 的连接过程就是这样从顶层到底层，按照调用次序一层层地调用相应层的连接函数实现的。

类似地，RDP 初始化和断开等也是按照类似的层次进行调用

初始化：

sec_init－> mcs_init－> iso_init－> tcp_init

断开连接：

rdp_disconnect－> sec_disconnect－> mcs_disconnect－> iso_disconnect－> tcp_disconnect

2.3. rdesktop 主程序执行过程

当 rdesktop 启动时 rdesktop.c 文件中的 main 函数将会执行。在 main 函数里分析命令行选项（用 getopt 函数，标准的 unix 程序），初始化图形用户接口，建立其它的一些东西（进程间通信，剪切板），然后尝试连接到命令行所指定的服务器。

连接过程通过调用 rdp.c 文件中的 rdp_connect 函数，此 rdp_connect 函数又会调用 sec_connect（secure.c）初始化一些数据，然后又继续调用 mcs_connect（mcs.c）。MCS 首先调用 iso_connect（iso.c）打开一个 ISO 连接然后协商其连接参数，包括传输图像、剪切板、声音和其它数据的通道。最后调用调用 tcp.c 文件中的函数来建立连接。

当建立连接之后，main 函数将创建一个窗口（用 xwin.c 中的 UI 函数）并且调用 rdp_main_loop（rdp.c），这是 rdesktop 程序的主循环。

在主循环中，rdesktop 尝试读取从服务器发送过来的数据包，当接收到包之后就立刻进行处理。读取数据包牵涉到各个层，程序通过调用 rdp_recv 来接收包，而 rdp_recv 又调用 sec_recv，sec_recv 调用 mcs_recv，mcs_recv 调用 iso_recv，iso_recv 调用 tcp_recv 接收到网络上的数据包后返回。这里值得注意的是，在 tcp.c 中接收到的数据结构是静态分配的，虽然每次接收到数据包后就会用 xrealloc（rdesktop.c）动态分配实际数据的大小。

所有数据包处理都是通过 STREAM 结构体来进行，STREAM 是在 parse.h 中定义的宏。

发送数据包和接收数据包的过程相似。当发送一个 RDP 数据包时，你要调用 rdp_init_data 函数来初始化一个 STREAM 结构体，并且确定数据包的最大长度。此函数调用 sec_init 设置标志和最大长度，并添加 RDP 头的长度到最大长度里作为一个参数。此过程会依次往下在 MCS，ISO 和 TCP 层执行，执行过程和 sec_init 相似。数据包初始化完成之后，数据就被填充到 STREAM 结构体中（STREAM 是在 parse.h 中定义的宏）。

2.4. STREAM 数据结构

我们知道，RDP 协议是一个网络协议，数据要在网络中传输，自然要把数据封装成一个网络数据包，然后通过网络发送出去。**在 rdesktop 中的源码中，有一个非常重要的数据结构，叫 STREAM，从网络接收的数据或向网络发送的数据包都通过 STREAM 进行,它在各层中都会被调用到。**rdesktop 把要发送的数据依次经过 RDP、SEC、MCS、ISO、TCP 各层的填充封装之后，再发送出去。rdesktop 从网络接收到的数据包也是有一个相类似的处理过程，对收到的 TCP 包的数据进行初期分解，按照不同的协议逐步将包中的 RDP 数据分离出来，为 RDP 数据的进一步分解做准备。　　在源码文件 parse.h 中，STREAM 结构体的定义如下所示：

```
typedef struct stream
{    unsigned char *p;
     unsigned char *end;
     unsigned char *data;
     unsigned int size;
     /* Offsets of various headers */
     unsigned char *iso_hdr;
     unsigned char *mcs_hdr;
     unsigned char *sec_hdr;
     unsigned char *rdp_hdr;
     unsigned char *channel_hdr;
} *STREAM;
```

STREAM 结构体中各个成员变量的作用如下：

p 指针：临时指针变量，主要用于指示数据包的指针位置。

data 指针：TCP/IP 数据的起始位置，是申请的一段内存，只当数据尺寸大于 size 时，进行 realloc 增大，不缩小。

size 指针：TCP/IP 数据的大小

end 指针：TCP/IP 数据的结束位置

```
//以上是 tcp 数据管理变量
iso_hdr 指针：TCP/IP 数据包中 iso 协议控制头的位置
mcs_hdr 指针：TCP/IP 数据包中 mcs 协议控制头的位置
sec_hdr 指针：TCP/IP 数据包中 sec 协议控制头的位置
rdp_hdr 指针：TCP/IP 数据包中 rdp 协议控制头的位置
channel_hdr 指针：TCP/IP 数据包中 channel 协议控制头的位置
```

下面是从 rdesktop 中截取了一段代码，我们用这段代码来说明程序是如何把想要发送的数据添加到 STREAM 里面的。

```
mcs_send_to_channel(STREAM s, uint16 channel)
{
    uint16 length;
    s_pop_layer(s, mcs_hdr);
    length = s->end - s->p - 8;
    length |= 0x8000;
    out_uint8(s, MCS_SDRQ );
    out_uint16(s, g_mcs_userid);
    out_uint16(s, channel);
    out_uint8(s, 0x70);
    out_uint16_be(s, length);
    iso_send(s);
}
```

上面代码是一个通道发送函数。既然是发送数据，必须要把想要发送的内容填充到 STREAM 里面才能发送出去。我们只关注粉红色的代码段（为了描述如何把数据封装到 STREAM 里面，这段代码已经作了修改）。可以看到 s 是一个 STREAM 变量，也就是说数据是要封装到 s 里面的，要发送的数据填充到 s->data 里面。

out_uint8(s, MCS_SDRQ)语句的表示将 MCS_SDRQ（这个变量占 8 位，也就是一个字节）这个变量的值封装到 s 里面。

紧接着 out_uint16(s, g_mcs_userid)会将 g_mcs_userid（占 16 位，即两字节）变量的值也封装到 s 里面。接下来的三个语句，依次将 channel、0x70、length 的值按顺序封装到 s 里面。执行完这几个语句以后，MCS_SDRQ、g_mcs_userid、channel、0x70、length 这几个数据的值在 s->data 中的位置如图 2.2 所示：

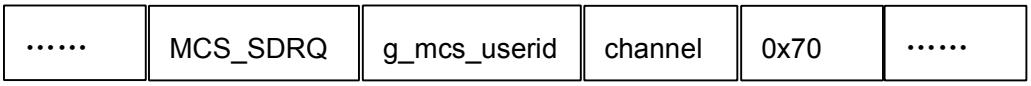

图 2.2 添加到 s->data 中的数据

对于接收到的数据包也有类似的处理，假如在 rdesktop 中有如下的代码：

```
in_uint16(s, channel);
in_uint16(s, length);
In_uint32(s,temp);
```

那么，上面这三个语句实现的是从 s->data 里面依次取出 2 字节数据给 channel，取出 2 字节数据给 length，最后取出 4 个字节给 temp，数据取出来之后就可以作后续的处理了。

其实像上面的 in_uint16 和 out_uint8 等，它们不是一个函数，而是一个带参数宏。从名称也可以大概猜出其用途，首先 in 和 out 分别代表往 STREAM 的 data 指针里添加数据和读取数据，uint 代表 unsigned int 类型，16 代表 16 位也即 2 个字节，同样 32 代表 4 个字节的数据。当你阅读源码的时候，你会发现这些宏定义都是通过 STREAM 结构体里的 p 指针，非常巧妙地实现 p 指针在 data 里的指向，从而实现数据的封装和抽取，因为数据就在 data 里面。要封装的数据是存到 STREAM 结构体的 data 变量量，要抽取的数据是从 data 里读出来。