FreeRdp工作流程

mayInteract 用来控制是否能控制被控端

hook掉shadow_input_register_callbacks函数，修改他参数input的MouseEvent

shadow_input_mouse_event 处做自动机，如果输入的x,y是连续满足特征的序列

其中x和y的范围是0-25535

功能设计：文件上传、命令执行、端口转发、隐匿模式

rdpShadowServer 关键成员

```
clients → ArrayList*
CertificateFile → char*
PrivateKeyFile → char*
listener → freerdp_listene
subsystem → rdpShadow
screen → rdpShadowScre
capture → rdpShadowCap
thread → shadow_server_
```

freerdp_listener 关键成员

```
PeerAccepted → shadow_client_a
Open → freerdp_listener_open
GetFileDescriptor → freerdp_lister
GetEventHandles → freerdp_lister
CheckFileDescriptor → freerdp_lis
Close → freerdp_listener_close
```

freerdp_peer 关键成员

```
sockfd → int
context → rdpShadowClient*
ContextExtra → rdpShadowServer*
ContextNew → shadow_client_context_new
ContextFree → shadow_client_context_free
Initialize → freerdp_peer_initialize
GetFileDescriptor → freerdp_peer_get_fds
GetEventHandle →freerdp_peer_get_event_ha
GetEventHandles →freerdp_peer_get_event_h
CheckFileDescriptor → freerdp_peer_check_fi
Close → freerdp_peer_close
Disconnect → freerdp_peer_disconnect
SendChannelData → freerdp_peer_send_chan
SendServerRedirection → freerdp_peer_send_
IsWriteBlocked → freerdp_peer_is_write_block
DrainOutputBuffer → freerdp_peer_drain_outp
HasMoreToRead → freerdp_peer_has_more_to
VirtualChannelOpen → freerdp_peer_virtual_cl
VirtualChannelClose → freerdp_peer_virtual_c
VirtualChannelWrite → freerdp_peer_virtual_cl
VirtualChannelGetData → freerdp_peer_virtua
VirtualChannelSetData → freerdp_peer_virtual
```

rdpShadowServer 的 thread 伪代码如下

```
listener→Open()
while(running) {
    event = listener→GetEventHandles()
    switch(event) {
        case xxx:
            running = false;
            break;
        default:
            listener→CheckFileDescriptor()
            break;
    }
}
listener→Close()
```

CheckFileDescriptor 伪代码如下

```
for( fd in sockfds) {
    peer_fd = _accept(fd)
    peer = freerdp_peer_new(peer_fd)
    PeerAccepted(listener, peer)
}
```

PeerAccepted 会启动线程，运行 **shadow_client_thread**

```
peer →settings = copy(server→settings)
freerdp_peer_context_new(peer)
Thread(shadow_client_thread, peer→context)
```

freerdp_peer_context_new 运行流程

input_new 运行流程

```
peer→context = new client
client→peer = peer
rdp = rdp_new(client)
peer → input = client→ input = rdp → input
peer → update = client→ update = rdp → update
peer → settings = client→ settings = rdp → settings = se
peer → autodetect = client→ autodetect = rdp → autode
update_register_server_callbacks(peer→update)
autodetect_register_server_callbacks(peer→autodetect)
transport_attach(rdp→transport, peer→sockfd)
rdp→transport→ReceiveCallback = peer_recv_callback
rdp→transport→ReceiveExtra = peer
```

```
rdpInput* input_new(rdpRdp* rdp)
{
  const wObject cb = { NULL, NULL, NULL, input_free_(
  rdpInput* input;
  input = (rdpInput*)calloc(1, sizeof(rdpInput));

  if (!input)
    return NULL;

  input→queue = MessageQueue_New(&cb);

  if (!input→queue)
  {
    free(input);
    return NULL;
  }

  return input;
}
```

rdp_new 运行流程

```
rdp→transport = transport_new(context);
rdp→license = license_new(rdp);
rdp→input = input_new(rdp);
rdp→update = update_new(rdp);
rdp→fastpath = fastpath_new(rdp);
rdp→nego = nego_new(rdp→transport);
rdp→mcs = mcs_new(rdp→transport);
rdp→redirection = redirection_new();
rdp→autodetect = autodetect_new();
rdp→heartbeat = heartbeat_new();
rdp→multitransport = multitransport_new();
rdp→bulk = bulk_new(context);
```

shadow_client_thread 运行流程

```
shadow_input_register_callbacks(peer→input);
```

shadow_client_activate的shadow_encoder_reset可以设置帧率

win_shadow_surface_copy是服务端发给客户端的屏幕数据

shw_client_thread

fastpath_recv_input_event_mouse读取鼠标输入

fastpath是关键点啊，通过他看一下数据是怎么处理的

fastpath中有注释： 快速通道（Fast - Path）有 15 位可用于长度信息，这将导致最大协议数据单元（PDU）大小为 0x8000。然而，实际上仅使用了 14 位。这一点在任何文档中都未提及，但看起来如果收到大小超过 0x3FFF 的快速通道数据包，大多数实现都会出错。

peer.c的peer_recv_pdu处理数据接收

shadow_client里面有DRDYNVC_STATE_READY

shadow_channels里面有shadow_client_channels_post_connect

grdp的鼠标事件不是fastpath，freerdp客户端是fastpath

input.c的input_recv_event是处理的非fastpath事件，即tkpt数据包

fastpath.c里面fastpath_recv_input_event

input.c里面input_recv_event

win_shadow里面SurfaceCopy

PDULayer
SEC
MCSClient
X224
TPKT
SocketLayer

pduLayer.transport发送PDU报文

对于tkpt数据包

MouseEvent(extends InputEvent)

uint16	uint16	uint16
PointerFlags	XPos	YPos

SlowPathInputEvent

uint32	uint16	int	byte[]
EventTime	MessageType	Size	SlowPathInputData
ignore	by {SlowPathInputData}	by {SlowPathInputData}.len	

ClientInputEventPDU(extends DataPDUData) 这里对应**input.c**的**input_recv**

uint16	uint16	{SlowPathInputEvent}
NumEvents	Pad20ctets	
by {SlowPathInputEvent}.num	ignore	

DataPDU(extends PDUMessage 前7个字段叫ShareDataHeader) 这里可以通过**WITH_DEBUG_RDP**控制日志，在**peer.c**的**peer_recv_data_pdu**

uint32	uint8	uint8	uint16	uint8	uint8	uint16
SharedId	Padding1	StreamId	UncompressedLength	PDUType2	CompressType	CompressLeng
ignore	ignore	used but ignored	by |DataPDUData|.len + 4	by |DataPDUData|.type2	used but ignored	ignore

PDU(前3个字段是ShareControlHeader) 这里对应**peer.c**的**peer_recv_tpkt_pdu**

uint16	uint16	uint16	|PDUMessage|
TotalLength	PDUType	PDUSource	Message
by |PDUMessage|.len + 6	by |PDUMessage|.type		

接收到的bitmap走的是fastpath，在pdu.go的**RecvFastPath**

fastpath.c的fastpath_recv_update，可以带上**WITH_DEBUG_RDP**

发送在update.c的update_send_bitmap_update

update_write_bitmap_update对应go的

`(f *FastPathBitmapUpdateDataPDU) Unpack`

实现额外的DataPDUData报文，设置PDUType2为原有的，而StreamId作为额外判断依据

PDUType2代表真实报文类型，StreamId表示是否为自定义报文

StreamId为0x05代表文件系列报文

FileTransferHead

uint16	uint16	byte[]	byte[]
FileNameLength	FilePathLength	FileName	FilePath

FileTransferStart(extends DataPDUData)文件传输开始报文 — PDUType2 = 0x36

|FileTransferHead|	uint32
	FileSize

FileTransferAbort(extends DataPDUData)文件传输中止报文 — PDUType2 = 0x27

|FileTransferHead|

FileTransferPacket(extends DataPDUData)文件传输内容报文 — PDUType2 = 0x21

|FileTransferHead|	uint32	uint32	byte[]
	StartIndex	SlicingSize	FileDataSlicing

FileTransferVerify(extends DataPDUData)文件传输校验报文 — PDUType2 = 0x02

|FileTransferHead|	uint32	uint32
	FileSize	CRC32

传输成功报文，传输失败报文。失败的话服务端删除已传输的。

FileTransferVerifyState 文件传输校验响应报文 — BakType = 0x02

|FileTransferHead|	uint8
	State
	全0 or 全1

图片隐写采用EMD算法，隐写到RGB信道的R，定义报文格式如下

uint16	uint16	uint32	[]byte	uint32
MagicCode	DataSize	BakType	Data	CRC32
0xdead	Max(200-8)			

EMD算法是4个二进制转为2个五进制。每两个像素隐写一位5进制信息。

对于40x40的RGB图片，仅取一个通道，共有1600个像素，可隐写800个五进制数据，转为2进制就是1600个，1600除8得到200Byte。

响应报文使用200Byte绰绰有余

做一个生产者消费者队列，verify的返回结果扔到队列，队列会定时取数据发送fastpath

StreamId为0x06代表控制系列报文

ControlStatePacket(extends DataPDUData)控制报文 — PDUType2 = 0x02

uint16	uint8	uint8
MagicCode1	State	MagicCode2
0x8135	全0 or 全1	0xb3

每个图片是64*64，四个像素隐写一个byte。

64*64可以隐写1024个byte，正常最大可以隐写1024个byte

StreamId为0x07代表命令行系列报文

CmdResetPacket(extends DataPDUData)命令行重置报文 — PDUType2 = 0x021

uint32
MagicCode1
0xdeadbeef

CmdOutPutResponse 命令行输出报文 — BakType = 0x04

uint16	[]byte
Length	Data
max 1000	

CmdInputPacket(extends DataPDUData)命令行输入报文 — PDUType2 = 0x027

uint16	[]byte
Length	Data
max 1000	

---

server端总计1161行代码

client端总计3179行代码