Fuzzing and Exploiting Virtual Channels in Microsoft Remote Desktop Protocol for Fun and Profit

Chun Sung Park
Yeongjin Jang
Seungjoo Kim*
Ki Taek Lee

* Corresponding Author
ABOUT US

• Chun Sung Park
  • Graduate student at SANE LAB, Korea University
  • Also a CTO of Diffense, Inc.

• Yeongjin Jang
  • Assistant Professor of Computer Science, Oregon State University

※ This work is partially supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2018-0-00532, Development of High-Assurance (≥EAL6) Secure Microkernel)
ABOUT US

• Seungjoo (Gabriel) Kim  (Corresponding Author, skim71@korea.ac.kr)
  • Professor of Graduate School of Information Security, Korea University

• Ki Taek Lee
  • Ph.D. candidate at SANE Lab, Korea University
  • Offensive security researcher and penetration tester at Samsung Research

※ This work is partially supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2018-0-00532, Development of High-Assurance(EAL6) Secure Microkernel)
AGENDA

• Motivation
• Background: Recon on the official MS RDP client and its protocol
• Finding Vulnerability Automatically: Build an RDP client fuzzer
• Applying the RDP Client Fuzzer
• Vulnerabilities
• RDP Heap Feng Shui
• Demo
• Future work & Conclusion
MOTIVATION

• Hackers are using some RDP servers to shadow their IP address for attacks
• Hacked windows servers in public are configured as an RDP server
  • Attackers launch attacks from this computer by logging into the RDP server
  • Victims can only see the IP address of the RDP server
MOTIVATION

• Allegedly being used by many North Korean Hackers
  • No proof, but from off-the-record discussions with security analysts

• Came up with a fun motivation:
  • Can we pwn RDP clients to launch attacks back to attackers?

![Diagram of RDP server with a North Korean flag and a devil icon]
MOTIVATION

• Disclaimer
  • We did not use this techniques in public (yes, it’s illegal)
  • We would like to share our journey for such a cool motivation in this presentation
Another Note

• Disclaimer
  • Some part of this research project cannot be disclosed by the restrictions applied to one of the speakers (NDA-related)
  • But, we will disclose as much as we can to entertain the audiences!
Recon on MS RDP client

- Need an understand the RDP protocol
- Requires two files to take a look at (via IDA Pro)
  - Mstsc.exe
  - Mstscax.dll
Recon on MS RDP client

• Additional (human-readable) resources: publicly available documentations
  • https://github.com/FreeRDP/FreeRDP/wiki/Reference-Documentation
  • 56 documentations about RDP specification is out there

Reference Documentation

Remote Desktop Protocol:

Windows Communication Protocols

[MS-RDPCR2]: Remote Desktop Protocol: Composited Remoting V2 Specification
[MS-RDPEA]: Remote Desktop Protocol: Audio Output Virtual Channel Extension
[MS-RDPEAI]: Remote Desktop Protocol: Audio Input Redirection Virtual Channel Extension
[MS-RDPECLIP]: Remote Desktop Protocol: Clipboard Virtual Channel Extension
[MS-RDPEDC]: Remote Desktop Protocol: Desktop Composition Virtual Channel Extension
[MS-RDPEEDYC]: Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension
[MS-RDPEECO]: Remote Desktop Protocol: Virtual Channel Echo Extension
Recon on MS RDP client

- We may also get hints for the RDP protocol from one of the open-source implementation: FreeRDP
  - [https://github.com/FreeRDP/FreeRDP](https://github.com/FreeRDP/FreeRDP)
  - Read the code, headers, data types, enums, etc. -> enables more efficient reversing
**Recon on MS RDP client**

- Latest Attack Case by CHECKPOINT
- Attacked the clipboard channel : Ctrl C + Ctrl V
  - A Path-traversal attack!
  - Poisoned RDP at BH USA 2019

- In contrast, our focus is at:
  - **Memory Corruption vulns** in the MS RDP Client that allows **Remote Code Execution**
Recon on MS RDP client

• Available resources to the RDP client

Remote Desktop clients

2018. 05. 07. • 읽는 데 2분 • 🦁 🦊


You can use a Microsoft Remote Desktop client to connect to a remote PC and your work resources from almost anywhere using just about any device. You can connect to your work PC and have access to all of your apps, files, and network resources as if you were sitting at your desk. You can leave apps open at work and then see those same apps at home - all by using the RD client.
Recon on MS RDP client

- How many (official) RDP Clients are there?
- 5 types by OS/App
  - mstsc.exe (native app)
  - Windows 10 app (AppContainer)
  - Android App
  - iOS App
  - macOS App
Recon on MS RDP client

• How many (official) RDP Clients are there?
  • 5 types by OS/App
    • mstsc.exe (native app)
    • Windows 10 app (AppContainer)
    • Android App
    • iOS App
    • macOS App

We assume all these implementation will share the same code base for the protocol
Our GOAL

• Find an exploitable memory corruption vulnerability in official RDP clients
• Popping shell from the RDP Client
  • Need to find the part of the program that handles ‘memory’ a lot
    • Intuition: more code for handling memory, more memory corruption vulnerabilities!
  • We purposely send malicious data from a compromised RDP Server to a client using virtual channels
What is Virtual Channel?

Remote Desktop Services virtual channels

05/31/2018 • 2 minutes to read • 

Virtual channels are software extensions that can be used to add functional enhancements to a Remote Desktop Services application. Examples of functional enhancements might include: support for special types of hardware, audio, or other additions to the core functionality provided by the Remote Desktop Services Remote Desktop Protocol (RDP). The RDP protocol provides multiplexed management of multiple virtual channels.
Virtual Channel Examples

• Default Channels

Internally, handing of such features includes protocol parsing logic, lots of heap memory allocation/free, etc., so it’s a great target for fuzzing!
Virtual Channel Protocol

- Open Server
- Open Channel
- Read Channel
- Write Channel
- Close Channel
- Close Server

**WTSOpenServerA function**

12/05/2018 • 2 minutes to read

Opens a handle to the specified Remote Desktop Session Host (RD Session Host) server.

**Syntax**

```
HANDLE WTSOpenServerA(  
    LPSTR pServerName  
);  
```
Virtual Channel Protocol
• Open Server
• **Open Channel**
• Read Channel
• Write Channel
• Close Channel
• Close Server

**WTSVirtualChannelOpen function**
12/05/2018 • 2 minutes to read

Opens a handle to the server end of a specified virtual channel.

This function is obsolete. Instead, use the **WTSVirtualChannelOpenEx** function.

**Syntax**

```cpp
HANDLE WTSVirtualChannelOpen(
    IN HANDLE hServer,
    IN DWORD SessionId,
    LPSTR pVirtualName
);
```

<table>
<thead>
<tr>
<th>C++</th>
<th>RDPSND</th>
<th>CLIPRDR</th>
<th>PRINTER</th>
<th>SMARTCARD</th>
<th>....</th>
</tr>
</thead>
</table>

@black hat events
Virtual Channel Protocol

• Open Server
• Open Channel
• **Read Channel**
• Write Channel
• Close Channel
• Close Server

**WTSVirtualChannelRead function**

12/05/2018 • 2 minutes to read

Reads data from the server end of a virtual channel.

`WTSVirtualChannelRead` reads the data written by a `VirtualChannelWrite` call at the client end of the virtual channel.

**Syntax**

```c++
BOOL WTSVirtualChannelRead(
    IN HANDLE hChannelHandle,
    IN ULONG TimeOut,
    PCHAR Buffer,
    IN ULONG BufferSize,
    PULONG pBytesRead
);
```
Virtual Channel Protocol

- Open Server
- Open Channel
- Read Channel
- **Write Channel**
- Close Channel
- Close Server

**WTSVirtualChannelWrite function**

12/05/2018 • 2 minutes to read

Writes data to the server end of a virtual channel.

**Syntax**

```cpp
BOOL WTSVirtualChannelWrite(
    IN HANDLE hChannelHandle,
    PCHAR Buffer,
    IN ULONG Length,
    PULONG pBytesWritten
);
```
Virtual Channel Protocol

- Open Server
- Open Channel
- Read Channel
- Write Channel
- Close Channel
- Close Server

WTSVirtualChannelClose function

12/05/2018 • 2 minutes to read

Closes an open virtual channel handle.

Syntax

```cpp
BOOL WTSVirtualChannelClose(
    IN HANDLE hChannelHandle
);
```
Virtual Channel Protocol

- Open Server
- Open Channel
- Read Channel
- Write Channel
- Close Channel
- Close Server

WTSCloseServer function

12/05/2018 • 2 minutes to read

Closes an open handle to a Remote Desktop Session Host (RD Session Host) server.

Syntax

```c++
void WTSCloseServer(
    IN HANDLE hServer
);
```
Virtual Channel Protocol

- Open Server
- Open Channel
- Read Channel
- Write Channel
- Close Channel
- Close Server

```c
hWTSSERVERHandle = WTSOpenServer("localhost");
hVirtualChannel = WTSVirtualChannelOpen(
    hWTSSERVERHandle,
    WTS_CURRENT_SESSION,
    "RDPSND"
);

buffer[0] = 0xa0;
buffer[1] = 0xa1;
buffer[2] = 0xa2;
buffer[3] = 0xa3;

BufferSize = 4;
bSuccess = WTSVirtualChannelWrite(hVirtualChannel, buffer, BufferSize, &bytesWritten);`
Virtual Channel Protocol

• Open Server
• Open Channel
• Read Channel
• Write Channel
• Close Channel
• Close Server

1. Run virtual channel write test program

```
C:\Windows\system32\cmd.exe
```

```
C:\Users\john\source\repos\channel_test\x64\Debug>channel_test.exe
WISVirtualChannelWrite written
Success!
```

2. Hooking mstscax!CChan::ChannelOnPacketReceived()

Attacker Controllable Data, will be parsed by the client
Attack Scenario via Manipulating the Virtual Channel

• From server to client using virtual channel
  • 1) Open server
  • 2) Open channel
  • 3) Write channel
  • 4) Send Malicious Data to Client
Finding Vulnerabilities Automatically: Build an RDP client fuzzer

• Requirements
  • Need to hook **Virtual Channels**
  • Need to work with a server-client model
  • Need to apply a blackbox fuzzing
  • On Windows
  • But we would like to enable coverage-guided fuzzing (like AFL)!
  • ...
Finding Vulnerabilities Automatically: Build an RDP client fuzzer

• Available coverage-guided fuzzing tool on Windows
  • WinAFL!

D:\Codes\winaf1\buildx86\Release>afl-fuzz.exe -i minset_test.dyn -o o2 -D D:\codes\DynamoRIO-Windows-6.2.0-2\bin32 -t 10000 -- -covtype edge -coverage_module test.exe -fuzz_iterations 5000 -target_module test.exe -target_method main -nargs 2 -- test.exe 00
Finding Vulnerabilities Automatically: Build an RDP client fuzzer

- **WinAFL**

- A fork of AFL for fuzzing Windows binaries
  - `afl-fuzz [afl options] -- [instrumentation options] -- target_cmd_line`
    - `afl-fuzz.exe -i in -o out -D C:\work\win afl\DynamoRIO\bin64 -t 20000 -- -coverage_module gdiplus.dll -coverage_module WindowsCodecs.dll -fuzz_iterations 5000 -target_module test_gdiplus.exe -target_offset 0x16e0 -nargs 2 -- test_gdiplus.exe @@`
Finding Vulnerabilities Automatically: Build an RDP client fuzzer

• WinAFL: slightly different with AFL
  
  • `afl-fuzz.exe -i in -o out -D C:\work\win afl\DynamoRIO\bin64 -t 20000 -- -- coverage_module gdiplus.dll -coverage_module WindowsCodecs.dll - fuzz_iterations 5000 -target_module test_gdiplus.exe -target_offset 0x16e0 -nargs 2 -- test_gdiplus.exe @@`

• Need to specify target module and offset to hook the function and measure the code coverage
Finding Vulnerabilities Automatically: Build an RDP client fuzzer

- WinAFL
  - It supports the following instrumentation modes:
    - Dynamic instrumentation using DynamoRIO
    - Hardware tracing using Intel PT
    - Static instrumentation via Syzygy
Finding Vulnerabilities Automatically: Build an RDP client fuzzer

• WinAFL
  • It supports the following instrumentation modes:

• Dynamic instrumentation using DynamoRIO
  • Drawback: slow
• Hardware tracing using Intel PT
• Static instrumentation via Syzygy
Finding Vulnerabilities Automatically: Build an RDP client fuzzer

- WinAFL
  - It supports the following instrumentation modes:
    - Dynamic instrumentation using DynamoRIO
    - Hardware tracing using Intel PT
      - Drawback: need a CPU that supports PT, and cannot run fuzzer in the VM
    - Static instrumentation via Syzygy
Finding Vulnerabilities Automatically: Build an RDP client fuzzer

- WinAFL
  - It supports the following instrumentation modes:
  - Dynamic instrumentation using DynamoRIO
  - Hardware tracing using Intel PT
  - Static instrumentation via Syzygy
    - Drawback: some restrictions...

There's some restriction to this:

- You binary should be a Win32 PE binary linked with the /PROFILE flag and statically linked to the CRT.
- If you are instrumenting a Chrome build, you might run out of memory if you are using a 32 bits version of Windows, we recommend that you use Windows 7/8 x64.
- Your binary should be compiled with level function linking enabled and buffer security checks disabled.
- The instrumenter requires to have the DIA SDK (msdia120.dll), it's installed by default with Visual Studio 2013 but we can't redistribute it. msdia120.dll is part of the C++ 2013 redistributable, but the path (C:\Program Files (x86)\Common Files\microsoft shared\VC) is not registered like with Visual Studio (C++). However, copying the msdia120.dll file to the instrumenter directory (or registering the path) should solve the problem.
Finding Vulnerabilities Automatically: Build an RDP client fuzzer

• We decided to work with DynamoRIO with in-app persistent mode

In App Persistence mode

This feature is a tweak for the traditional "target function" approach and aims to loosen the requirements of the target function to do both reading an input file and processing the input file.

In some applications it's quite challenging to find a target function that with a simple execution redirection won't break global states and will do both reading and processing of inputs.

This mode assumes that the target application will actually loop the target function by itself, and will handle properly its global state For example a udp server handling packets or a js interpreter running inside a while loop.

This mode works as following:

1. Your target runs until hitting the target function.
2. The afl server starts instrumenting the target.
3. Your target runs until hitting the target function again.
4. The afl server stops instrumenting current cycle and starts a new one.
Finding Vulnerabilities Automatically: Build an RDP client fuzzer

• The fuzzer architecture for fuzzing RDP client (a modified version of WinAFL)

• Client
  • run winafl with mstsc.exe and hook mstscax.dll (for data receiving function)
    • Check code coverage, mutate seed files
    • Make a Cur_input file
    • Send Cur_input file to server

• Server
  • Run a program that
    • Receive Cur_input file from WinAFL
    • The content of cur_input will be sent through the virtual channel

Loop
Finding Vulnerabilities Automatically: Build an RDP client fuzzer

- Original Architecture of WinAFL
Finding Vulnerabilities Automatically: Build an RDP client fuzzer

• Modified WinAFL for fuzzing the RDP client
Applying the RDP Client Fuzzer

• Fuzzing mstsc.exe On Windows With WinAFL via Virtual Channels in RDP

• First target: **RDPSND**
  • A channel enabled by **default** by mstsc.exe
  • **One-way** communication as audio playback is run by server and played in the client
  • Very simple protocol

• Note: other channels (Clipboard, etc.) are two-way channels
Applying the RDP Client Fuzzer

- Fuzzing mstsc.exe On Windows With WinAFL
- RDPSND : Audio Output Virtual Channel
Applying the RDP Client Fuzzer

• Fuzzing mstsc.exe On Windows With WinAFL
  • RDPSND : Audio Output Virtual Channel

The Server Audio Formats and Version PDU is a PDU used by the server to send version information and a list of supported audio formats to the client. This PDU MUST be sent using virtual channels.

| Header | dwFlags | dwVolume | dwPitch | wDGramPort | wNumberOfFormats | cLastBlockConfirmed | wVersion | sndFormats (variable) | ...
|--------|---------|----------|---------|------------|------------------|--------------------|----------|----------------------|------|

2.2.1 RDPSND PDU Header (SNDPROLOG)

The RDPSND PDU header is present in many audio PDUs. It is used to identify the PDU type, specify the length of the PDU, and convey message flags.
Applying the RDP Client Fuzzer

• Fuzzing mstsc.exe On Windows With WinAFL
Applying the RDP Client Fuzzer

• Fuzzing mstsc.exe On Windows With WinAFL
  • RDPSND : Audio Output Virtual Channel
  • How to get a seed file for fuzzer?
    • Hooking and Logging

msgType
msgType (1 byte): An 8-bit unsigned integer that specifies the type of audio PDU that follows the BodySize field.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>Close PDU</td>
</tr>
<tr>
<td>0x02</td>
<td>Waveinfo PDU</td>
</tr>
<tr>
<td>0x03</td>
<td>Volume PDU</td>
</tr>
<tr>
<td>0x04</td>
<td>Pitch PDU</td>
</tr>
<tr>
<td>0x05</td>
<td>Wave Confirm PDU</td>
</tr>
<tr>
<td>0x06</td>
<td>Training PDU or Training Confirm PDU</td>
</tr>
<tr>
<td>0x07</td>
<td>Server Audio Formats and Version PDU or Client Audio Formats and Version PDU</td>
</tr>
</tbody>
</table>
Applying the RDP Client Fuzzer

• Fuzzing mstsc.exe On Windows With WinAFL
  • RDPSND : Audio Output Virtual Channel

```c
switch (msgType) {
  case SNDC_FORMATS:
    status = rdpsnd_recv_server_audio_formats_pdu(rdpsnd, s);
    break;

  case SNDC_TRAINING:
    status = rdpsnd_recv_training_pdu(rdpsnd, s);
    break;

  case SNDC_WAVE:
    status = rdpsnd_recv_wave_info_pdu(rdpsnd, s, BodySize);
    break;

  case SNDC_CLOSE:
    rdpsnd_recv_close_pdu(rdpsnd);
    break;

  case SNDC_SETVOLUME:
    status = rdpsnd_recv_volume_pdu(rdpsnd, s);
    break;
}
```

<table>
<thead>
<tr>
<th>msgType (1 byte)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNDC_CLOSE</td>
<td>Close PDU</td>
</tr>
<tr>
<td>0x01</td>
<td></td>
</tr>
<tr>
<td>SNDC_WAVE</td>
<td>Waveinfo PDU</td>
</tr>
<tr>
<td>0x02</td>
<td></td>
</tr>
<tr>
<td>SNDC_SETVOLUME</td>
<td>Volume PDU</td>
</tr>
<tr>
<td>0x03</td>
<td></td>
</tr>
<tr>
<td>SNDC_SETPITCH</td>
<td>Pitch PDU</td>
</tr>
<tr>
<td>0x04</td>
<td></td>
</tr>
<tr>
<td>SNDC_WAVECONFIRM</td>
<td>Wave Confirm PDU</td>
</tr>
<tr>
<td>0x05</td>
<td></td>
</tr>
<tr>
<td>SNDC_TRAINING</td>
<td>Training PDU or Training Confirm PDU</td>
</tr>
<tr>
<td>0x06</td>
<td></td>
</tr>
<tr>
<td>SNDC_FORMATS</td>
<td>Server Audio Formats and Version PDU or Client Audio Formats and Version PDU</td>
</tr>
<tr>
<td>0x07</td>
<td></td>
</tr>
</tbody>
</table>
Applying the RDP Client Fuzzer

• Fuzzing mstsc.exe On Windows With WinAFL

• Next targets
  • Other Channels
    • Drdynvc
    • Printer
    • Smartcard
    • ...
Applying the RDP Client Fuzzer

• Demonstration: RDP Client Fuzzer
  • Running on Windows 7
  • 1 Core
  • 2G Memory
  • 2 VMs are required
    • One for RDP Client
    • One for RDP Server
Running the RDP Client Fuzzer

• Fuzzing setup
  • Both the RDP Client and the RDP Server runs in each VM
  • Running on Windows 7
  • 1 Core
  • 2G Memory
  • 2 VMs are required
    • One for RDP Client
    • One for RDP Server
  • The first vulnerability was found within 2 hours of running the RDP Client Fuzzer!
    • We keep reporting vulnerabilities to MS
Detecting Heap Vulnerabilities

• We may use PageHeap; allocate each heap object in a new page
  • `gflags.exe /p /enable mstsc.exe /full`
  • **Slow**, but this will generate crash if any heap error happens during fuzzing..

Full page heap block structure

[Diagram showing the structure of a Page Heap Block]

MSDN Article: The Structure of a Page Heap Block
## VULNERABILITY & EXPLOIT

### • RDP Client vulnerabilities on Windows in 2019

<table>
<thead>
<tr>
<th>Category</th>
<th>Title</th>
<th>Author</th>
<th>Date</th>
<th>Ref</th>
<th>Threat No</th>
</tr>
</thead>
</table>

Report by Checkpoint
## VULNERABILITY & EXPLOIT

### • RDP Client vulnerabilities on Windows in 2019

<table>
<thead>
<tr>
<th>Category</th>
<th>Title</th>
<th>Author</th>
<th>Date</th>
<th>Ref</th>
<th>Threat No</th>
</tr>
</thead>
</table>

No comment on: CVE-2019-1108; guess who reported this
# RDP Client vulnerabilities on Windows in 2019

<table>
<thead>
<tr>
<th>Category</th>
<th>Title</th>
<th>Author</th>
<th>Date</th>
<th>Ref</th>
<th>Threat No</th>
</tr>
</thead>
</table>
## VULNERABILITY & EXPLOIT

**• RDP Client vulnerabilities on Windows in 2019**

<table>
<thead>
<tr>
<th>Category</th>
<th>Title</th>
<th>Author</th>
<th>Date</th>
<th>Ref</th>
<th>Threat No</th>
</tr>
</thead>
</table>
# Vulnerability & Exploit

## RDP Client vulnerabilities on Windows in 2019

<table>
<thead>
<tr>
<th>Category</th>
<th>Title</th>
<th>Author</th>
<th>Date</th>
<th>Ref</th>
<th>Threat No</th>
</tr>
</thead>
</table>

For the next, we will describe how to exploit the vulnerability found by our RDP Client Fuzzer, by combining information leak and remote code execution vulnerabilities...

No comment on: CVE-2019-1108; guess who reported this
Vulnerabilities and Exploit

• We will break the ASLR in the RDP client

• We will achieve RCE in the RDP client

• The RCE vulnerability is a heap vulnerability
  • We need to manipulate heap objects
    • Need heap feng shui
RDP Heap Feng Shui

• Exploiting a DRDYNVC Channel
• DRDYNVC is a dedicated channel for delivering dynamic data

1.3.1.1 Encapsulation in the DRDYNVC Static Virtual Channel

The following diagram illustrates the wire-level encapsulation when a DVC is embedded inside the dedicated static virtual channel named DRDYNVC.

![Diagram of encapsulation](image)

Figure 1: Static virtual channel objects

This is a Windows implementation detail and does not limit the definition and the description of the Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension. Any transport that has similar characteristics can be used to support a DVC implementation. The Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension makes use of the following features of a static virtual channel:
RDP Heap Feng Shui

- In the DRDYNVC Channel

```c
#define CREATE_REQUEST_PDU 0x01
#define DATA_FIRST_PDU 0x02
#define DATA_PDU 0x03
#define CLOSE_REQUEST_PDU 0x04
#define CAPABILITY_REQUEST_PDU 0x05

switch (Cmd) {
    case CAPABILITY_REQUEST_PDU:
        return drdynvc_process_capability_request(drdynvc, Sp, cbChId, s);
    case CREATE_REQUEST_PDU:
        return drdynvc_process_create_request(drdynvc, Sp, cbChId, s);
    case DATA_FIRST_PDU:
        return drdynvc_process_data_first(drdynvc, Sp, cbChId, s);
    case DATA_PDU:
        return drdynvc_process_data(drdynvc, Sp, cbChId, s);
    case CLOSE_REQUEST_PDU:
        return drdynvc_process_close_request(drdynvc, Sp, cbChId, s);
}
```

In FreeRDP Client Source code

```c
Stream_Read_UINT8(s, value);
Cmd = (value & 0xf0) >> 4;
Sp = (value & 0xc0) >> 2;
cbChId = (value & 0x03) >> 0;
```
RDP Heap Feng Shui

- Heap Control Primitives in the DRDYNVC Channel
  - DATA_FIRST_PDU
  - Calling buf = malloc(size)
RDP Heap Feng Shui

• Heap Control Primitives in the DRDYNVC Channel
  • DATA_FIRST_PDU
  • Calling buf = malloc(size)

```c
case DATA_FIRST_PDU:
    return drdynvc_process_data_first(drdynvc, Sp, cbChId, s);
```

- drdynvc_process_data_first()
- dvcman_receive_channel_data_first()
- StreamPool_Take()
- Malloc(size)
- dvcman_receive_channel_data()
RDP Heap Feng Shui

- Heap Control Primitives in the DRDYNVC Channel
  - DATA_PDU
  - Calling memcpy(buf, input, size)

```c
static UINT drdynvc_process_data(drdynvc, Sp, int cbChId, wStream s)
{
    UINT32 ChannelId;
    if (Stream_GetRemainingLength(s) < drdynvc_cblen_to_bytes(cbChId))
        return ERROR_INVALID_DATA;
    ChannelId = drdynvc_read_variable_uint(s, cbChId);
    WLog_Print(drdynvc->log, WLOG_TRACE, "process_data: Sp=\d cbChId=\d, ChannelId=\n"
                cbChId, ChannelId);
    return dvcman_receive_channel_data(drdynvc, drdynvc->channel_mrg, ChannelId, s);
}
```

```c
if (channel->dvc_data)
{
    /* Fragmented data */
    if (StreamGetPosition(channel->dvc_data) + dataSize
```

```c
Stream_COPY(data, channel->dvc_data, dataSize);
```
RDP Heap Feng Shui

• Heap Control Primitives in the DRDYNVC Channel
  • DATA_PDU
  • Calling `memcpy(buf,input,size)`

```c
case DATA_PDU:
    return drdynvc_process_data(drdynvc, Sp, cbChId, s);
```

- `dvcman_receive_channel_data()`
- `memcpy(input,Buf,Size)`
- `Stream_Copy()`
RDP Heap Feng Shui

- Heap Control Primitives in the DRDYNVIC Channel
  - Close Channel
  - Calling `free(buf)`

```c
if (channel->dvc_data)
    Stream_Release(channel->dvc_data);
```

```c
void dvcman_channel_free()
```

Exploitation

• Combining Information Leak & RCE vulnerabilities
• This part has been removed due to the restriction of one of our speakers
  • NDA...

• But, after having an arbitrary control over malloc(), memcpy(), and free()
  • We can do feng shui to spray the data and manipulate heap objects to achieve arbitrary code execution
  • Fairly standard after following RDP Heap Feng Shui primitives

Please do not be disappointed, we will show you a demo of the full-chain RCE exploit!
DEMO

• Exploiting an Information Leak vulnerability (guess which CVE it is!)
• Use of Uninitialized memory; affected to:
  • Mstsc.exe
  • Windows 10 app
  • Android App
  • iOS App
  • macOS App
• All five RDP clients (MS official)!

Our assumption on sharing the code base among those seems true! Fuzz mstsc.exe and pwn four more!
DEMO

• Exploiting an Information Leak vulnerability (guess which CVE it is!)
• Use of Uninitialized memory; affected to:
  • Mstsc.exe
  • Windows 10 app
  • Android App
  • iOS App
  • macOS App
• All five RDP clients (MS official)!

Our assumption on sharing the code base among those seems true! Fuzz mstsc.exe and pwn four more!
DEMO

• **Breaking ASLR** (guess which CVE it is!)
  • Mstsc.exe on windows 7
  • We will demonstrate how we can break the ASLR of the client machine by launching a memory leak attack.
DEMO

• Remote Code Execution
  • Mstsc.exe on windows 7
  • Using RDP Heap Feng Shui
    • Leak memory to break ASLR
    • Allocate heap objects in a pretty good construction
    • Hijacking a vtable of an object and launch Return-oriented Programming (ROP)
      • Pops a calc.exe!
FUTURE WORK

• Fuzzing on other channels
  • Drdynvc
  • Printer
  • Smartcard
  • …

• Need to handle two-way communication
  • Modifying the program at the server is required
FUTURE WORK & Conclusion

• We have learned **coverage-guided fuzzing** to
  • Windows Application
  • Server-client model
  • Applications with no source code available
  • Without modifying the binary program (no requirement for syzygy)
FUTURE WORK & Conclusion

• We have learned **coverage-guided fuzzing** to
  • Windows Application
  • Server-client model
  • Applications with no source code available
  • Without modifying the binary program (no requirement for syzygy)

• **Don’t be afraid** the application runs over server-client and does not accept file input
THANKS

• Q&A
DEMO

• RDP CLIENT FUZZER
<table>
<thead>
<tr>
<th>Name</th>
<th>Date modified</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>afl_docs</td>
<td>7/28/2019 1:19 AM</td>
<td>File folder</td>
<td></td>
</tr>
<tr>
<td>afl_post_library</td>
<td>7/28/2019 1:19 AM</td>
<td>File folder</td>
<td></td>
</tr>
<tr>
<td>bin32</td>
<td>7/28/2019 1:19 AM</td>
<td>File folder</td>
<td></td>
</tr>
<tr>
<td>bin64</td>
<td>7/28/2019 1:19 AM</td>
<td>File folder</td>
<td></td>
</tr>
<tr>
<td>build64</td>
<td>8/1/2019 8:45 PM</td>
<td>File folder</td>
<td></td>
</tr>
<tr>
<td>screenshots</td>
<td>7/28/2019 1:19 AM</td>
<td>File folder</td>
<td></td>
</tr>
<tr>
<td>testcases</td>
<td>7/28/2019 1:19 AM</td>
<td>File folder</td>
<td></td>
</tr>
<tr>
<td>third_party</td>
<td>7/28/2019 1:19 AM</td>
<td>File folder</td>
<td></td>
</tr>
<tr>
<td>gitmodules</td>
<td>7/28/2019 1:19 AM</td>
<td>File folder</td>
<td></td>
</tr>
<tr>
<td>afl-analyze.c</td>
<td>7/28/2019 1:19 AM</td>
<td>C File</td>
<td>32 KB</td>
</tr>
<tr>
<td>afl-fuzz.c</td>
<td>8/3/2019 8:00 AM</td>
<td>C File</td>
<td>205 KB</td>
</tr>
<tr>
<td>afl-showmap.c</td>
<td>7/28/2019 1:19 AM</td>
<td>C File</td>
<td>26 KB</td>
</tr>
<tr>
<td>afl-staticinstr.c</td>
<td>7/28/2019 1:19 AM</td>
<td>C File</td>
<td>22 KB</td>
</tr>
<tr>
<td>afl-staticinstr.h</td>
<td>7/28/2019 1:19 AM</td>
<td>C File</td>
<td>7 KB</td>
</tr>
<tr>
<td>afl-tmin.c</td>
<td>7/28/2019 1:19 AM</td>
<td>C File</td>
<td>34 KB</td>
</tr>
<tr>
<td>alloc-ini.h</td>
<td>7/28/2019 1:19 AM</td>
<td>H File</td>
<td>13 KB</td>
</tr>
<tr>
<td>ChangeLog</td>
<td>7/28/2019 1:19 AM</td>
<td>File</td>
<td>8 KB</td>
</tr>
<tr>
<td>CMakelists.txt</td>
<td>7/28/2019 1:19 AM</td>
<td>Text Document</td>
<td>4 KB</td>
</tr>
<tr>
<td>config.h</td>
<td>7/28/2019 1:19 AM</td>
<td>H File</td>
<td>12 KB</td>
</tr>
<tr>
<td>CONTRIBUTING.md</td>
<td>7/28/2019 1:19 AM</td>
<td>MD File</td>
<td>2 KB</td>
</tr>
<tr>
<td>custom_net_fuzzer.c</td>
<td>7/28/2019 1:19 AM</td>
<td>C File</td>
<td>8 KB</td>
</tr>
<tr>
<td>custom_net_fuzzer.def</td>
<td>7/28/2019 1:19 AM</td>
<td>DEF File</td>
<td>1 KB</td>
</tr>
<tr>
<td>custom_winaf.server.c</td>
<td>7/28/2019 1:19 AM</td>
<td>C File</td>
<td>6 KB</td>
</tr>
<tr>
<td>custom_winaf_server.def</td>
<td>7/28/2019 1:19 AM</td>
<td>DEF File</td>
<td>1 KB</td>
</tr>
</tbody>
</table>
THANKS

• Q&A