

# LPC & ALPC interfaces

Windows privilege escalation

#### Thomas Garnier SkyRecon Systems

Recon 2008 – 05/23/2008

#### Overview

SkyRecon

- Introduction
- LPC interface details
- MS08-002: LSASS privilege escalation
- Demo: LSASS exploitation
- Protection against LPC privilege escalation
- ALPC architecture improvement
- MS07-066: ALPC kernel code execution
- Demo: ALPC exploitation
- Protecting the Windows kernel
- Conclusion



# Introduction

#### Introduce LPC & ALPC

SkyRecon®

- What is the LPC interface ?
  - Stands for "Local Procedure Call"
  - Created for the Windows NT kernel for Windows subsystem
  - Undocumented kernel component
  - Provides local communication across processes
- What is the ALPC interface ?
  - Stands for "Advanced Local Procedure Call"
  - Added in Windows Vista (still undocumented)
  - Supports old LPC functions
  - Redesign of LPC architecture and features

#### Why LPC is interesting ?



- Many SYSTEM processes provide public LPC interfaces
- Hidden in classical Windows API functions
- Local transportation for RPC and OLE
- Share mapped section across processes
  - Available on almost all LPC interfaces (default)
  - Improved privilege escalation reliability
  - <u>WLSI</u> by Cesar Cerudo
- LPC is now well documented on the internet

#### Why ALPC is interesting ?



- Shares interesting points with LPC
- Design concerns
  - Important new component of the Windows Vista kernel
  - Compatibility with LPC interfaces
  - Performance improvement
- Security concern
  - Modification on remote section mapping
  - Security Development Lifecycle (SDL)
  - LPC interface was secure after multiple security patch



# LPC interface details

#### LPC interface details

- SkyRecon\*
- Classical communication architecture (Server / Client)
- LPC works with a named object named a Port
  - Specified during port creation and connection
  - Supports ACL restriction by security descriptor
- Incoming connection can be discarded
- Message based communication
  - Queue mechanism a HANDLE per client on server (optional)
  - Multiple message types (connection, request, reply...)

#### Past LPC vulnerabilities

- SkyRecon\*
- LPC interface was secured though many patches
  - MS00-003 Spoofed LPC Port Request
  - **MS00-070** Multiple LPC and LPC Port Vulnerabilities
  - MS03-031 Cumulative Patch for Microsoft SQL Server
  - **MS04-044** Vulnerabilities in Windows Kernel and LSASS
  - MS07-029 Windows DNS RPC Interface
- First vulnerabilities allowed message spoofing / sniffing
- LPC design issues

#### LPC research



- There are many reasons for looking at LPC interfaces
- Understand LPC design
- Possible restriction of public interfaces
- Block remote section mapping
- Privilege escalation techniques
  - Understand basics
  - Think about new protection layers against it



# LSASS privilege escalation

#### LSASS LPC interface



- The Local Security Authority Subsystem Service (LSASS) provides services for local and domain users
  - Critical system component
  - Handles user authentication (access to SAM database), user and group privileges, password policies ...
- Isasrv.dll manages "\LsaAuthenticationPort" port
  - Public port available with a guest account
  - Almost all LSASS features are provided by this interface
  - Implements a LPC dispatch table

#### LSASS dispatch table



LpcDispatchTable dd offset LpcLsaLookupPackage@4

DATA XREF: DispatchAPI(x)+2D<sup>†</sup>r : DispatchAPIDirect(x)+CATr : LpcLsaLookupPackage(x) dd offset LpcLsaLogonUser@4 ; LpcLsaLogonUser(x) dd offset LpcLsaCallPackage@4 ; LpcLsaCallPackage(x) dd offset LpcLsaDeregisterLogonProcess@4 ; LpcLsaDeregisterLogonProcess(x) dd Ø dd offset LpcGetBinding@4 ; LpcGetBinding(x) dd offset LpcSetSession@4 ; LpcSetSession(x) dd offset LpcFindPackage@4 ; LpcFindPackage(x) dd offset LpcEnumPackages@4 ; LpcEnumPackages(x) dd offset LpcAcquireCreds@4 ; LpcAcquireCreds(x) dd offset LpcEstablishCreds@4 ; LpcEstablishCreds(x) dd offset LpcFreeCredHandle@4 : LpcFreeCredHandle(x) dd offset LpcInitContext@4 ; LpcInitContext(x) dd offset LpcAcceptContext@4 ; LpcAcceptContext(x) dd offset LpcApplyToken@4 ; LpcApplyToken(x) dd offset LpcDeleteContext@4 ; LpcDeleteContext(x) dd offset LpcQueryPackage@4 : LpcQueryPackage(x) dd offset LpcGetUserInfo@4 ; LpcGetUserInfo(x) dd offset LpcDeleteCreds@4 ; LpcDeleteCreds(x) dd offset LpcDeleteCreds@4 ; LpcDeleteCreds(x) dd offset LpcDeleteCreds@4 ; LpcDeleteCreds(x) dd offset LpcQueryCredAttributes@4 ; LpcQueryCredAttributes(x) dd offset LpcAddPackage@4 ; LpcAddPackage(x) dd offset LpcDeletePackage@4 ; LpcDeletePackage(x) dd offset LpcEfsGenerateKey@4 ; LpcEfsGenerateKey(x) dd offset LpcEfsGenerateDirEfs@4 ; LpcEfsGenerateDirEfs(x) dd offset LpcEfsDecryptFek@4 ; LpcEfsDecryptFek(x) dd offset LpcEfsGenerateSessionKey@4 : LpcEfsGenerateSessionKey(x) dd offset LpcCallback@4 ; LpcCallback(x) dd offset LpcQueryContextAttributes@4 ; LpcQueryContextAttributes(x) dd offset LpcLsaPolicyChangeNotify@4 ; LpcLsaPolicyChangeNotify(x) dd offset LpcGetUserName@4 ; LpcGetUserName(x) dd offset LpcAddCredentials@4 ; LpcAddCredentials(x) dd offset LpcEnumLogonSessions@4 : LpcEnumLogonSessions(x) dd offset LpcGetLogonSessionData@4 ; LpcGetLogonSessionData(x) dd offset LpcSetContextAttributes@4 ; LpcSetContextAttributes(x) dd offset LpcLookupAccountName@4 ; LpcLookupAccountName(x) dd offset LpcLookupAccountSid@4 ; LpcLookupAccountSid(x)

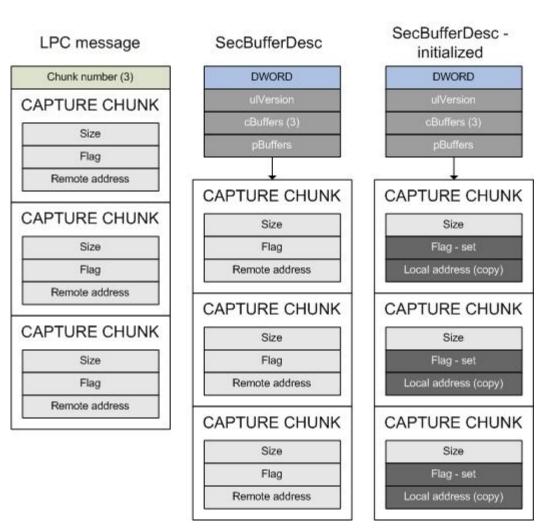
#### LSASS - Remote buffer system



- The *LpcInitContext* and *LpcAcceptContext* functions have their own remote buffer system
- The *LsapCaptureBuffers* function captures buffer list
- The *MapTokenBuffer* function mirrors remote data
- The LsapUncaptureBuffers function liberates allocated buffers
- A vulnerability exists in the way unintialized resources are liberated in the *LpcInitContext* function

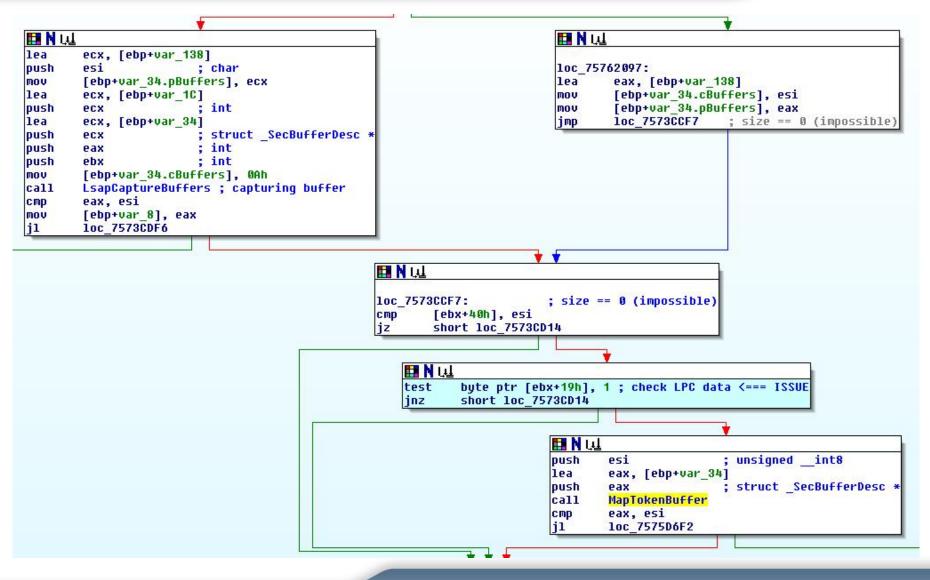
#### LSASS - SecBufferDesc





#### Vulnerable assembly





#### **RtlFreeHeap exploitation**

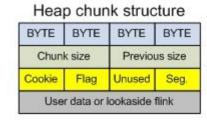


- Frees a crafted chunk in a remote mapped section.
- RtlFreeHeap function algorithm (XP SP2)
  - Verifies chunk integrity (Cookie / Flags / Alignment)
  - Looks at previous and next chunk for coalescing
  - It goes in lookaside table if:
    - » Lookaside list support is enabled
    - » Size < 1024 bytes
    - » Lookaside entry < 3 entries.
- Client process changes lookaside next entry pointer
- Next allocation of the size returns a custom pointer

#### Bypass cookie verification

- Bruteforce heap cookie is possible on a static address
  - Cookie verification algorithm: ((ChunkAddr >> 3) ^ (ChunkCookie) ^ (HeapCookie)) == 0
  - Only 256 possibilities
  - An invalid cookie does not stop the process.

# Bruteforce while testdata is unchanged



BYTE	BYTE	BYTE	BYTE
Target size		0	
XX	01	01	01

Bruteforce cookie (0 to 255)
 Modified data when free worked

#### 2) Corrupt lookaside

SkyRecon<sup>®</sup>

BYTE	BYTE	BYTE	BYTE	
Target size		0		
XX	00	01	01	
	Target	address		

3) Final					
BYTE	BYTE	BYTE	BYTE		
Target size		Ó			
XX	01	01	01		
NULL					

Chunk flags : 00 : Free 01 : Busy

#### **Overwrite target**



- Overwrites any part of the memory from 8 bytes to 1024 bytes
- The Data Execution Prevention (DEP) activation restriction
- The LSASS LPC dispatch table contains an empty entry



- Uses pattern matching to untouch other entries
- First dword must be a zero (protects lookaside integrity)
- Specific context (message data not far)

#### **Control flow redirection**

- SkyRecon\*
- Windows XP SP2, the LPC dispatch table call context:
  - First argument and EDI register point to the message
  - 0x18 first bytes of this buffer are not fully controlled
- Context register can change between module versions (service pack, language pack)
- Getting stack control with ntdll.dll assembly
- Deactivate DEP protection
- Jump in remote mapped section



# **Demo – LSASS exploitation**



# **Protecting LPC interfaces**

#### **Restrict mapped section**



- Improves privilege local escalation reliability
- No publicly known public interface uses it
- Used by some private kernel LPC interfaces
  - o \SeLsaCommandPort
  - o \XactSrvLpcPort
- Black list model
- Restriction based on right level (with a whitelist)

#### **DEP** hardening



- DEP protection contributes to operating system security
- In Windows Vista, kernel32.dll module has a SetProcessDEPPolicy function
  - The only argument changes DEP status (FALSE is deactivated)
  - Easier exploitation (ret-to-libc)
  - Microsoft considers DEP status modification as a feature
- Disable DEP deactivation is not clever
- Distinguish a legitimate deactivation

#### Userland heap security



- The *RtlFreeHeap* function allows exploitation
- Windows Vista improvement
  - The heap chunk is xored with a random value
  - On some configurations an invalid chunk stops the process (default is 64 bit platforms)
- Many different types of protection can be created
  - Disallow freeing of a buffer which failed previous attempts
  - Filter returned pointer from the *RtlAllocateHeap* function
  - Performance issues can be important

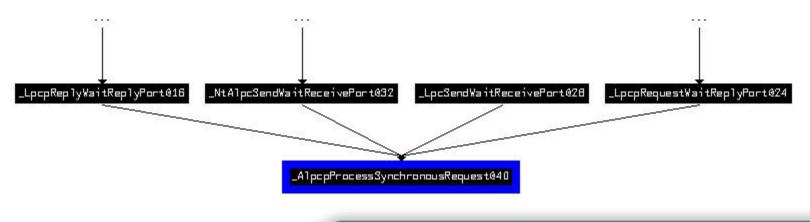


# **ALPC interface details**

#### ALPC interface details

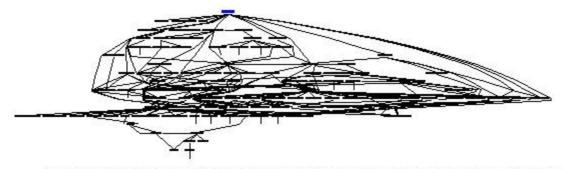


- New version of the LPC interface added in Windows Vista
  - The old LPC code no longer exists
  - ALPC and LPC shared a same code base (code modularity)
  - Supports I/O completion port (thread organization mechanism)
  - Userland server message treatment improved
  - Global performance improvement (asynchronous)

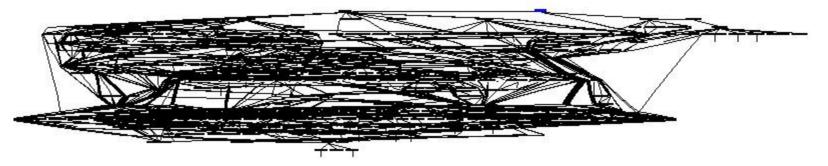


#### NtRequestPort crossref





NtRequestPort crossref - from (4 level) on Windows XP SP2 - 130 nodes - 374 edges segments - 402 crossing



NtRequestPort crossref - from (4 level) on Windows Vista SPO - 333 nodes - 1818 edges segments - 7717 crossing

#### New interface functions



- This new kernel component has 21 syscall functions starting with "NtAlpc"
- Message send and receive is done by a single function called NtAlpcSendWaitReceivePort
- Where LPC used 4 different functions
- Totally new functions
  - Open sender thread / process
  - Create section representation
  - Security context
  - Resource reserve

#### Message function



• The send and receive function:

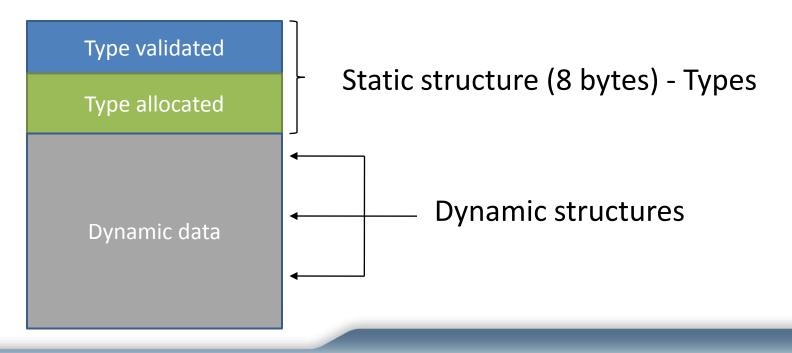
NTSTATUS NTAPI NtAlpcSendWaitReceivePort( HANDLE PortHandle, DWORD SendFlags, // Same as connection flag PLPC\_MESSAGE SendMessage OPTIONAL, PVOID InMessageBuffer OPTIONAL, PLPC\_MESSAGE ReceiveBuffer OPTIONAL, PULONG ReceiveBufferSize OPTIONAL, PVOID OutMessageBuffer OPTIONAL, PLARGE\_INTEGER Timeout OPTIONAL);

- The SendMessage and ReceiveBuffer arguments are optional depending on if you want to send or receive or both.
- The *InMessageBuffer* and *OutMessageBuffer* refers to action sent with a message

#### Message buffer system

SkyRecon\*

- Dynamic structure system
  - Contains multiple structures
  - Structures access is made though dedicated function
  - Compatible across operating system versions



#### Remote mapping steps



- Call *NtAlpcCreatePortSection* function
  - Submits a section or automatic section creation
  - Links the section with submitted port return and handle
- Create a message buffer
  - Include ALPC\_MESSAGE\_FLAG\_VIEW type
  - Set AlpcSectionHandle field to port section handle
- Call NtAlpcCreateSectionView function
  - Submit the message buffer view pointer
  - Initialized data and finalize kernel objects
- Use final message buffer

#### Remote mapping mitigation



- ALPC remote mapping update not default
  - Appropriate server message buffer
  - There is no remote mapping address returned
  - A section cannot be mapped twice during the same connection
- ALPC Message buffer architecture weirdness
  - Nothing indicates if remote mapping worked
  - ALPC connection message buffer
  - Disconnection does not unmap section (spray attack)



# ALPC kernel code execution

#### Resource reserve



- The resource reserve is a new feature of ALPC interface
  - A message object linked with a resource reserve object
  - This message object is unassociated with any process.
- New function syscall to create or destroy a resource reserve
  - NtAlpcCreateResourceReserve
  - NtAlpcDeleteResourceReserve
- Guessed feature no wild examples

#### Vulnerable assembly



```
eax, [ebp+var 20] ; will contain kernel message pointer
lea
push
       eax
       [ebp+var 30]
push
       [ebp+var 34]
                     ; <== messageid
push
push
       ebx
       @AlpcpLookupMessage@16 ; retrieve our kernel message
call
       [ebp+arg 8], eax
mov
       eax, eax
test
j1
       loc 5C5A6B
; Some check which always pass
       eax, [ebp+var 20]
mov
       ebx, [eax+3Ch] ; <=== no NULL check for ALPC server object
mov
       esi, [ebx+8] ; acces violation /!\ (control ESI value)
mov
       byte ptr [ebp+arg 8+3], cl
mov
       eax, [esi-10h]
lea
       [ebp+var 24], eax
mov
push
       11h
       ecx
pop
       edx, eax
mov
       eax, eax
xor
lock cmpxchg [edx], ecx
                             ; temporary DWORD overwrite with 0
test
       eax, eax
       short loc 5C56AA
                             ; old value was 0 ?
jz
       ecx, edx
mov
```

call @ExfAcquirePushLockShared@4 ; made overwritting permanent

### **NULL deference exploitation**

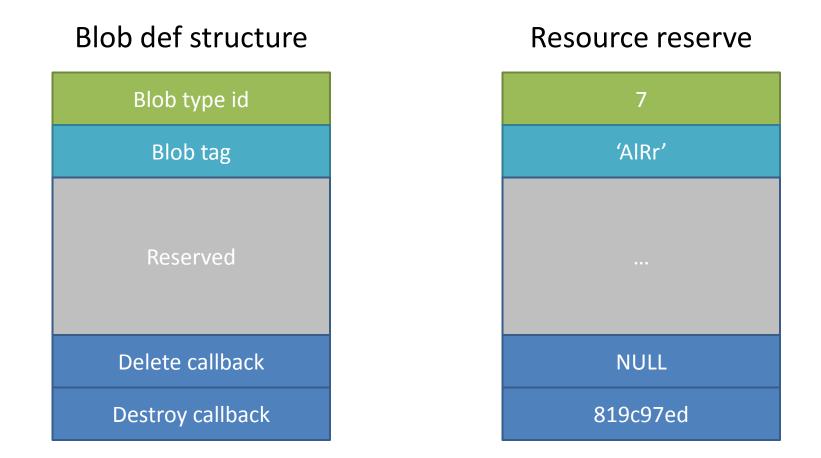
- Kernel NULL deference
  - NULL pointer is in kernel address space in a controlled process
  - Userland control used data by allocate NULL page
- Possible with *NtAllocateVirtualMemory* function Example:

AllocateAddr = (PVOID) sizeof(DWORD); nStatus = NtAllocateVirtualMemory((HANDLE)-1, &AllocateAddr, 0, &AllocateLength, MEM\_RESERVE | MEM\_COMMIT | MEM\_TOP\_DOWN, PAGE\_EXECUTE\_READWRITE);

SkyRecon

#### Targeting resource reserve





Unaligned delete callback: 0xed000000 *ExfAcquirePushLockShared* exploitation



# **Demo** – ALPC exploitation



# Protecting Windows kernel

#### Kernel local privilege escalation



- The next exploitation landscape
  - Kernel code does not have any protection
  - NULL deference is as important as an overflow (more stable)
- Kernel protection is harder
  - A single mistake crashes the system
  - Windows kernel was not built for third party protection
- Two basic protection approaches
  - Software against common attack vector
  - Hardware monitoring

#### Software protection

- Look at common attack vector
- Modifies operating system behavior
- Denied NULL page allocation
  - The system should not use it anyway?
  - In fact the system uses it a lot
  - Some packers could use it too
- Hardened kernel pool
  - Major change between 2000 and XP
  - Internal management structures unexported



#### Hardware monitoring



- Security by Virtualization
  - Easier to describe than to create
  - The more system is monitored, the more it will slow down
  - The best choice needs more research
- Apply PaX KERNEXEC recipes for Windows kernel
  - Separation between user mode and kernel mode address space
  - Kernel safe concept not applicable for Windows



# Conclusion

#### Conclusion



- There are still vulnerabilities in sub-system components
  - Unusual vulnerabilities
  - Windows Vista improved its code base and robustness
- Windows Vista privilege escalation
  - Userland components are much safer
  - The Windows kernel module is more secure than others
- Serious kernel protection would need operating system design progress



# Thank you ! Questions ?