LPC & ALPC interfaces
Windows privilege escalation

Thomas Garnier
SkyRecon Systems

Recon 2008 – 05/23/2008
Overview

- 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
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
  - WLSI by Cesar Cerudo
- LPC is now well documented on the internet
Why ALPC is interesting?

• Shares interesting points with LPC

• Design concerns
  o Important new component of the Windows Vista kernel
  o Compatibility with LPC interfaces
  o Performance improvement

• Security concern
  o Modification on remote section mapping
  o Security Development Lifecycle (SDL)
  o LPC interface was secure after multiple security patch
LPC interface details
LPC interface details

- 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...)

SkRecon®
Past LPC vulnerabilities

• LPC interface was secured though many patches
  o **MS00-003** - Spoofed LPC Port Request
  o **MS00-070** - Multiple LPC and LPC Port Vulnerabilities
  o **MS03-031** - Cumulative Patch for Microsoft SQL Server
  o **MS04-044** - Vulnerabilities in Windows Kernel and LSASS
  o **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
  o Understand basics
  o Think about new protection layers against it
LSASS privilege escalation
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 ...

lsasrv.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)+20Tr
; DispatchAPIDirect(x)+CATr
; LpcLsaLookupPackage(x)

dd offset _LpcLsaLogonUser@4 ; LpcLsaLogonUser(x)
dd offset _LpcLsaCallPackage@4 ; LpcLsaCallPackage(x)
dd offset _LpcLsaDeregisterLogonProcess@4 ; LpcLsaDeregisterLogonProcess(x)

dd 0

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 _LpcGetCreds@4 ; LpcGetCreds(x)

dd offset _LpcDeleteCreds@4 ; LpcDeleteCreds(x)

dd offset _LpcQueryCredsAttributes@4 ; LpcQueryCredsAttributes(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 _LpcEfsDecryptFk@4 ; LpcEfsDecryptFk(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 \textit{LpclInitContext} and \textit{LpcAcceptContext} functions have their own remote buffer system
- The \textit{LsapCaptureBuffers} function captures buffer list
- The \textit{MapTokenBuffer} function mirrors remote data
- The \textit{LsapUncaptureBuffers} function liberates allocated buffers
- A vulnerability exists in the way uninitialized resources are liberated in the \textit{LpclInitContext} function
LSASS - SecBufferDesc

LPC message

<table>
<thead>
<tr>
<th>Chunk number (3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAPTURE CHUNK</td>
</tr>
<tr>
<td>Size</td>
</tr>
<tr>
<td>Flag</td>
</tr>
<tr>
<td>Remote address</td>
</tr>
</tbody>
</table>

SecBufferDesc

<table>
<thead>
<tr>
<th>DWORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>ufVersion</td>
</tr>
<tr>
<td>cBuffers (3)</td>
</tr>
<tr>
<td>pBuffer</td>
</tr>
</tbody>
</table>

SecBufferDesc - initialized

<table>
<thead>
<tr>
<th>DWORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>ufVersion</td>
</tr>
<tr>
<td>cBuffers (3)</td>
</tr>
<tr>
<td>pBuffer</td>
</tr>
</tbody>
</table>

Capture Chunk

<table>
<thead>
<tr>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag</td>
</tr>
<tr>
<td>Remote address</td>
</tr>
</tbody>
</table>

Capture Chunk

<table>
<thead>
<tr>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag</td>
</tr>
<tr>
<td>Remote address</td>
</tr>
</tbody>
</table>

Capture Chunk

<table>
<thead>
<tr>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag</td>
</tr>
<tr>
<td>Remote address</td>
</tr>
</tbody>
</table>

Capture Chunk

<table>
<thead>
<tr>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag - set</td>
</tr>
<tr>
<td>Local address (copy)</td>
</tr>
</tbody>
</table>

Capture Chunk

<table>
<thead>
<tr>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag - set</td>
</tr>
<tr>
<td>Local address (copy)</td>
</tr>
</tbody>
</table>

Capture Chunk

<table>
<thead>
<tr>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag - set</td>
</tr>
<tr>
<td>Local address (copy)</td>
</tr>
</tbody>
</table>
Vulnerable assembly

```assembly
lea ecx, [ebp+var_138]
push esi ; char
mov [ebp+var_34.pBuffers], ecx
lea ecx, [ebp+var_1C]
push ecx ; int
lea ecx, [ebp+var_34]
push ecx ; struct _SecBufferDesc *
push eax ; int
push ebx ; int
mov [ebp+var_34.pBuffers], 0Ah
call LsapCaptureBuffers ; capturing buffer
cmp eax, esi
mov [ebp+var_8], eax
j1 loc_7573CDF6

loc_7573CCF7: ; size == 0 (impossible)
cmp [ebx+40h], esi
jz short loc_7573CD14

test byte ptr [ebx+19h], 1 ; check LPC data <<< ISSUE
jnz short loc_7573CD14

push esi ; unsigned __int8
lea eax, [ebp+var_34]
push eax ; struct _SecBufferDesc *
call MapTokenBuffer
cmp eax, esi
j1 loc_7575D6F2
```
RtlFreeHeap exploitation

• Frees a crafted chunk in a remote mapped section.
• RtlFreeHeap function algorithm (XP SP2)
  o Verifies chunk integrity (Cookie / Flags / Alignment)
  o Looks at previous and next chunk for coalescing
  o 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

- Brute force heap cookie is possible on a static address
  - Cookie verification algorithm:
    \[((\text{ChunkAddr} \gg 3) \ ^ \ (\text{ChunkCookie}) \ ^ \ (\text{HeapCookie})) == 0\]
  - Only 256 possibilities
  - An invalid cookie does not stop the process.

Brute force while test data is unchanged
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

```
LpcDispatchTable dd offset _LpcLsaLookupPackage@4
    ; DATA XREF: DispatchAPI(x)+2DTr
    ; DispatchAPIDirect(x)+CATr
    ; LpcLsaLookupPackage(x)
    dd offset _LpcLsaLogonUser@4 ; LpcLsaLogonUser(x)
    dd offset _LpcLsaCallPackage@4 ; LpcLsaCallPackage(x)
    dd offset _LpcLsaDeregisterLogonProcess@4 ; LpcLsaDeregisterLogonProcess(x)
    dd 0
    dd offset _LpcGetBinding@4 ; LpcGetBinding(x)
```

- 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

- 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
  - `\SeLsaCommandPort`
  - `\XactSrvLpcPort`
- Black list model
- Restriction based on right level (with a whitelist)
• DEP protection contributes to operating system security
• In Windows Vista, kernel32.dll module has a *SetProcessDEPPolicy* function
  o The only argument changes DEP status (FALSE is deactivated)
  o Easier exploitation (ret-to-libc)
  o 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 - from (4 level) on Windows XP SP2 - 130 nodes - 374 edges segments - 402 crossing

NtRequestPort crossref - from (4 level) on Windows Vista SP0 - 333 nodes - 1818 edges segments - 7717 crossing
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
The send and receive function:

```c
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

• Dynamic structure system
  o Contains multiple structures
  o Structures access is made through dedicated function
  o Compatible across operating system versions

Static structure (8 bytes) - Types

Type validated
Type allocated

Dynamic data

Dynamic structures
Remote mapping steps

• Call `NtAlpcCreatePortSection` function
  o Submits a section or automatic section creation
  o Links the section with submitted port return and handle

• Create a message buffer
  o Include `ALPC_MESSAGE_FLAG_VIEW` type
  o Set `AlpcSectionHandle` field to port section handle

• Call `NtAlpcCreateSectionView` function
  o Submit the message buffer view pointer
  o Initialized data and finalize kernel objects

• Use final message buffer
Remote mapping mitigation

• ALPC remote mapping update – not default
  o Appropriate server message buffer
  o There is no remote mapping address returned
  o A section cannot be mapped twice during the same connection

• ALPC Message buffer architecture weirdness
  o Nothing indicates if remote mapping worked
  o ALPC connection message buffer
  o 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
  - \texttt{NtAlpcCreateResourceReserve}
  - \texttt{NtAlpcDeleteResourceReserve}
- Guessed feature – no wild examples
Vulnerable assembly

```assembly
lea  eax, [ebp+var_20]    ; will contain kernel message pointer
push eax
push [ebp+var_30]
push [ebp+var_34]         ; <= messageid
push ebx
call @AlpcpLookupMessage@16 ; retrieve our kernel message
mov [ebp+arg_8], eax
test eax, eax
jl loc_5C5A6B

; Some check which always pass

mov  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
lea  eax, [esi-10h]
mov  [ebp+var_24], eax
push 11h
pop ecx
mov edx, eax
xor eax, eax
lock cmpxchg [edx], ecx   ; temporary DWORD overwrite with 0
test eax, eax
jz short loc_5C56AA      ; old value was 0 ?
mov ecx, edx
call @ExfAcquirePushLockShared@4 ; made overwritting permanent
```
Kernel NULL deference

- NULL pointer is in kernel address space in a controlled process
- Userland control used data by allocate NULL page

Possible with \texttt{NtAllocateVirtualMemory} function

Example:

```c
AllocateAddr = (PVOID) sizeof(DWORD);
nStatus = NtAllocateVirtualMemory((HANDLE)-1, &AllocateAddr, 0, 
&AllocateLength, MEM_RESERVE | MEM_COMMIT | 
MEM_TOP_DOWN, PAGE_EXECUTE_READWRITE);
```
Targeting resource reserve

Blob def structure

- Blob type id
- Blob tag
- Reserved
- Delete callback
- Destroy callback

Resource reserve

- 7
- ‘AIr\r’
- NULL
- 819c97ed

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
  o The system should not use it anyway?
  o In fact the system uses it a lot
  o Some packers could use it too
• Hardened kernel pool
  o Major change between 2000 and XP
  o Internal management structures unexported
Hardware monitoring

• Security by Virtualization
  o Easier to describe than to create
  o The more system is monitored, the more it will slow down
  o The best choice needs more research

• Apply PaX KERNEXEC recipes for Windows kernel
  o Separation between user mode and kernel mode address space
  o Kernel safe concept not applicable for Windows
Conclusion
Conclusion

• There are still vulnerabilities in sub-system components
  o Unusual vulnerabilities
  o Windows Vista improved its code base and robustness

• Windows Vista privilege escalation
  o Userland components are much safer
  o The Windows kernel module is more secure than others

• Serious kernel protection would need operating system design progress
Thank you! Questions?