Windows Security

Interprocess Communication

Adrian Coleșa

Universitatea Tehnică din Cluj-Napoca
Computer Science Department

12 august 2014
The purpose of this lecture
Outline

1. Windows IPC Security
   - The Redirector
   - Impersonation
2. Window Messaging
3. Pipes
4. Mailslots
5. Remote Procedure Calls
   - RPC Connections
   - RPC Transports
   - Microsoft Interface Definition Language (MIDL)
   - Application Configuration Files (ACF)
   - RPC Servers
   - Impersonation in RPC
Universal Naming Convention

- UNC network path: “\server\share\path”
- server could be: an IP, a NETBIOS name, a qualified DNS name
- server name “.” is for the local host
connection to any remote machine generates a set of session credentials for it, stored in the logon session

a login session maintains only one set of credentials per remote system
SMB Relay Attack

- when connecting to another system, the server presents the client with a random challenge value
- the client responds with a message authentication code (MAC) incorporating the password hash and the challenge value
- this way LAN Manager (LM) and NT LAN Manager (NTLM) authentication avoid presenting the password hash to a potentially malicious server
- problem: server identity is never verified
SMB Relay Attack (cont.)

- vulnerable to a type of a man-in-the-middle attack: SMB Relay or SMB proxy
  - causes the victim (e.g. by emails) to establish a SMB connection to an attacker-controlled system
  - initiates a connection to a target system and acts as a proxy between the victim and the target
  - after the challenge exchange is completed, the attacker is connected to the target server with the victim’s credentials
Purpose and Contents
Windows IPC Security
Window Messaging
Pipes
Mailslots
Remote Procedure Calls
Bibliography

The Redirector
Impersonation

Outline

1 Windows IPC Security
   • The Redirector
   • Impersonation

2 Window Messaging
3 Pipes
4 Mailslots
5 Remote Procedure Calls
   • RPC Connections
   • RPC Transports
   • Microsoft Interface Definition Language (MIDL)
   • Application Configuration Files (ACF)
   • RPC Servers
   • Impersonation in RPC
   • Context Handles and State
   • Threading in RPC
Impersonation Levels

- Impersonation allows credentials to be transferred automatically to processes in another session on the same or remote machine.
- One of the foundational components of Windows single sign-on (SSO) mechanism.
- Allow a client to restrict the degree to which an IPC server can use the client’s credentials.
- Levels:
  - `SecurityAnonymous`
  - `SecurityIdentification`
Impersonation Levels (cont.)

- **SecurityImpersonation**
- **SecurityDelegation**

  usually specified as a parameter in IPC connection functions
SeImpersonatePrivilege

- a required privilege for impersonating another user
- a normal user does not have it by default
- granted to the built-in service accounts
Purpose and Contents
Windows IPC Security
Window Messaging
Pipes
Mailslots
Remote Procedure Calls
Bibliography

Outline

1. Windows IPC Security
   - The Redirector
   - Impersonation

2. Window Messaging

3. Pipes

4. Mailslots

5. Remote Procedure Calls
   - RPC Connections
   - RPC Transports
   - Microsoft Interface Definition Language (MIDL)
   - Application Configuration Files (ACF)
   - RPC Servers
   - Impersonation in RPC
   - Context Handles and State
   - Threading in RPC

Secure Coding Course
Windows Security
Window Stations Object

- the primary method of isolating GUI-based communication
- contains essential GUI information, like
  - a private atom table (a shared collection of strings)
  - a clipboard
  - windows
  - one or more desktop objects
- each logon is associated with a single window station
- processes can be moved between window stations, assuming the associated tokens have adequate privileges
- *Winsta0*: the single window station for keyboard, mouse, and the primary display
all services running under the network service account share a single window station ("Service-0x0-3e6$") and desktop

all services running under a local service account share a separate desktop and window station ("Service-0x0-3e5$")

DACL on a window station is quite strict: limits access to essentially the system account and the owning user

for **Winsta0** an ACE for user’s SID is added to the DACL at logon and removed at logoff
Window Stations Object (cont.)

- processes started in the contexts other than the window station’s owner inherit an open handle to the window station from the parent process
The Desktop Object

- a securable UI object that functions as a display surface for attached threads
- every thread on the system is associated to a single desktop
- desktops exist as objects inside a window station
- *Winsta0* contains three desktop objects
  - default
  - Winlogon
  - screen server
The Desktop Object (cont.)

- access control on a desktop determines which users can manipulate the display surface
- desktop does not affect processing of window messages
Window Messages

- UI windows receive events through the use of window messages with the following structure:

  ```c
  typedef struct {
    HWND hwnd;
    UINT message;
    WPARAM wParam;
    LPARAM lParam;
    DWORD time;
    POINT pt;
  } MSG, *PMSG;
  ```

- OS delivers messages to windows in a FIFO manner generated by system events (e.g. mouse movements, key presses)
there are four essential steps in creating a functional windowed application

- create a `WindowProc()` function to handle messages
- define a class that associate the `WindowProc` to a window type
- create an instance of the `Window` class
- create a message-processing loop
Window Messages (cont.)

```c
int MainWindowProc(HWND hWnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    switch (iMsg) {
    case WM_CREATE:
        return 0;
    case WM_DESTROY:
        return PostQuitMessage(0);
    default:
        return DefWindowProc(hWnd, iMsg, wParam, lParam);
    }
}

BOOL InitClass(HINSTANCE hInst)
{
    WNDCLASSEX wc;

    ZeroMemory(&wc, sizeof(wc));
    wc.hInstance = hInst;
    wc.lpszClassName = "Main";
    wc.lpfnWndProc = (WNDPROC) MainWindowProc;
    wc.cbSize = sizeof(WNDCLASSEX);
}```
return RegisterClassEx(&wc);
}

int APIENTRY WinMain(HINSTANCE hInst, HINSTANCE hPrev,
                       LPSTR lpCmdLine, int nCmdShow)
{
    WINDOW hwnd;
    InitClass(hInst);

    hwnd = CreateWindow("Main", "Main", 0,0,0,0,0,0, HWND_MESSAGE, 0);

    while (GetMessage(&msg, 0, 0, 0) &
            GetMessage(&msg, 0, 0, 0) != -1) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return msg.wParam;
}
Window Messages (cont.)

- `CreateWindow` creates a message-only window that is never displayed, but only handles window messages.
- Function `SendMessage` could be used to send a window message to a window whose handle is available to a process.
there is no method to restrict and verify a message source
attackers must have access to a window station before they can send messages
the original shatter attack (privilege escalation)
  sends a “WM_PASTE” message to a privileged process with the message pump on the same window station
  this allows the attacker to place shell code in the address space of the privileged process
  the attack is completed by sending a “WM_TIMER” message that includes the address of the injected code
Shatter Attacks (cont.)

- many other messages like "WM_TIMER" could also be used similarly
- the root of the problem: a privileged process (e.g. a service) cannot safely interact with a potentially hostile desktop
- code audit: identify situations that cause a privileged service to interact with a normal user desktop
Dynamic Data Exchange (DDE)

- a legacy form of IPC that exchange data by using a combination of window messages and shared memory
- two ways
  - requires handling “WM_DDE_*” window messages with `PackDDEkParam()` and `UnpackDDEIParam()` functions
  - uses DDE Management Library (DDEML) API
- was a common form of IPC in earlier versions of Windows
- now deprecated
Dynamic Data Exchange (DDE) (cont.)

- has no real security impact when used to establish communication between processes with the same security context
- problems
  - supports communication between processes with different user security contexts on a shared window station
  - supports exchange data over the network by using file shares
  - supports impersonation of clients in a DDE communication
Terminal Sessions

- Windows Terminal Services (WTS) provides the capability of a single Windows system to host multiple interactive user sessions.
- Terminal sessions place additional restrictions on the interaction between processes in different sessions.
- Each terminal session has a unique Winsta0 associated with it.
- Objects are distinguished between sessions by using “Global\” and “Local\” namespaces prefixes.
Outline

1. Windows IPC Security
   - The Redirector
   - Impersonation
2. Window Messaging
3. Pipes
4. Mailslots
5. Remote Procedure Calls
   - RPC Connections
   - RPC Transports
   - Microsoft Interface Definition Language (MIDL)
   - Application Configuration Files (ACF)
   - RPC Servers
   - Impersonation in RPC
   - Context Handles and State
   - Threading in RPC
Pipe Permissions

- anonymous pipes
  - uni-directional
  - used between threads of a single process and between parent and child processes

- named pipes
  - can be referred to by arbitrary processes, even remotely
  - multi-directional
  - work in a client-server architecture (one server and one or more clients)

- all pipes are securable objects
- permissions
Pipe Permissions (cont.)

- PIPE_ACCESS_DUPLEX
- PIPE_ACCESS_INBOUND
- PIPE_ACCESS_OUTBOUND
Named Pipe Creation

- function `CreateNamedPipe()`

```c
HANDLE CreateNamedPipe(LPCTSTR lpName, DWORD dwOpenMode, DWORD dwPipeMode,
                        DWORD nMaxInstances, DWORD nOutBufferSize,
                        DWORD nInBufferSize, DWORD nDefaultTimeout,
                        LPSECURITY_ATTRIBUTES lpSecurityAttributes)
```

- `dwOpenMode`
  - PIPE_ACCESS_DUPLEX, PIPE_ACCESS_INBOUND,
  - PIPE_ACCESS_OUTBOUND,
  - FILE_FLAG_FIRST_PIPE_INSTANCE,
  - FILE_FLAG_WRITE_THROUGH,
  - FILE_FLAG_OVERLAPPED

- `dwPipeMode`
  - PIPE_TYPE_BYTE, PIPE_TYPE_MESSAGE
just sending a message on a pipe could be done using `CallNamedPipe()` function
Impersonation in Pipes

- A named pipe can impersonate the credentials of client servers that connect to it.
- Function `ImpersonateNamedPipeClient()`
- The calling thread impersonates the context of the last message read from the pipe.
  - If the connection is local, impersonation fails unless data has first been read from and written to the pipe.
  - If the client is remote, impersonation might succeed because messages are transferred in establishing the connection.
Impersonation in Pipes (cont.)

- in both cases, it is best to make sure the pipe is read from before impersonation is attempted
- clients can restrict the degree to which a server can impersonate them by specifying an impersonation level in `CreateFile()`
- impersonation flags in the `dwFlagsAndAttributes` parameter of `CreateFile()`, when “SECURITY_SQOS_PRESENT” is used

  - SECURITY_ANONYMOUS, SECURITY_IDENTIFICATION,
  - SECURITY_IMPERSONATION, SECURITY_DELEGATION,
  - SECURITY_EFFECTIVE_ONLY,
  - SECURITY_CONTEXT_TRACKING
Impersonation in Pipes (cont.)

- example of a correct code, protecting itself from being impersonation by the server

```c
hPipe = CreateFile("\\\.\\pipe\MyPipe", GENERIC_ALL, 0, NULL, OPEN_EXISTING,
SECURITY_SQOS_PRESENT | SECURITY_IDENTIFICATION, NULL);
```

- example of a vulnerable code omitting to check the result of impersonation function
Impersonation in Pipes (cont.)

```c
for (; ;) {
    rc = ReadFile(hPipe, bufferm BUFSIZE, &bytes, NULL);

    switch (buffer[0]) {
        case REQUEST_FILE:
            extract_filename(buffer, bytes, fname);
            ImpersonateNamedPipeClient(hPipe);
            write_file_to_pipe(hPipe, fname);
            RevertToSelf();
            break;
    }
}
```

- If `ImpersonateNamedPipeClient()` fails, the server (privileged application) write the file with its own privileges and permission rights.
Pipe Squatting

- it is possible for named pipes
- developers must be careful in deciding how to create and open a named pipe
- code review
  - implications for servers that are vulnerable to name squatting
  - implications for clients that are vulnerable to name squatting
Pipe Squatting for Servers

- when fails to check if the pipe has already been created
- when creates a pool of pipes and uses `ConnectNamedPipe()` to service multiple connections
- when creating a single-instance pipe using `CreateFile()`, squatting vulnerability could occur the same way it does for files
- example of vulnerable code, not using `FILE_FLAG_FIRST_PIPE_INSTANCE`

```c
hPipe = CreateNamedPipe("\\\.\pipe\MyPipe", PIPE_ACCESS_DUPLEX, PIPE_TYPE_BYTE, PIPE_INLIMITED_INSTANCES, 1024, 1024, NWPWAIT_USE_DEFAULT_WAIT, psd);
```
attacker can create and connect to a pipe named “MyPipe” before the vulnerable application

- clients connect to the attacker’s pipe
Pipe Squatting for Clients

- they can unintentionally connect to a malicious pipe
- introduction of `SeImpersonatePrivilege` eliminated this type of vulnerability
Outline

1. Windows IPC Security
   - The Redirector
   - Impersonation
2. Window Messaging
3. Pipes
4. Mailslots
5. Remote Procedure Calls
   - RPC Connections
   - RPC Transports
   - Microsoft Interface Definition Language (MIDL)
   - Application Configuration Files (ACF)
   - RPC Servers
   - Impersonation in RPC
   - Context Handles and State
   - Threading in RPC
Mailslot Permissions

- mailslots are neither connection-oriented nor bidirectional
- clients just send messages to the server
- clients never read from mailslots
- do not have a unique set of access rights: use the standard file access rights
Mailslot Squatting

- is not possible the same way as for other named objects
- they only have a creation function, `CreateMailslot()`, which fails if a mailslot of the same name already exists
- a client could send a message to a server it does not intend to
Outline

1. Windows IPC Security
   - The Redirector
   - Impersonation

2. Window Messaging

3. Pipes

4. Mailslots

5. Remote Procedure Calls
   - RPC Connections
   - RPC Transports
   - Microsoft Interface Definition Language (MIDL)
   - Application Configuration Files (ACF)
   - RPC Servers
   - Impersonation in RPC
   - Context Handles and State
   - Threading in RPC
RPC Connections

- Windows uses DCE RPC
- Before a client can make a RPC, it needs to create a binding
- Binding = an application-level connection between client and server
- Binding handles
- Endpoint mapper
Outline

1. Windows IPC Security
   - The Redirector
   - Impersonation

2. Window Messaging

3. Pipes

4. Mailslots

5. Remote Procedure Calls
   - RPC Connections
   - RPC Transports
   - Microsoft Interface Definition Language (MIDL)
   - Application Configuration Files (ACF)
   - RPC Servers
   - Impersonation in RPC
   - Context Handles and State
   - Threading in RPC
NCACN

- NCACN = network computing architecture connection oriented
  - ncacn_nb_tcp
  - ncacn_nb_ipx
  - ncacn_nb_nb
  - ncacn_ip_tcp
  - ncacn_np
  - ncacn_http
NCADG

- NCADG = network computing architecture datagram protocol
- connectionless
- ncadg_ip_udp
- ncadg_ipx
NCALRPC

- local RPC, when both server and client are on the same machine
- ncalrpc
Outline

1. Windows IPC Security
   - The Redirector
   - Impersonation
2. Window Messaging
3. Pipes
4. Mailslots
5. Remote Procedure Calls
   - RPC Connections
   - RPC Transports
   - Microsoft Interface Definition Language (MIDL)
   - Application Configuration Files (ACF)
   - RPC Servers
   - Impersonation in RPC
   - Context Handles and State
   - Threading in RPC
Definition

- MIDL has a C-like structure
- IDL Interface
- IDL definition body
- compiler features
IDL File Structure. IDL Interface Header

- specifies a set of attributes
  
  `attribute_name(Attribute_arguments)`

- example

  ```
  [ 
    `uuid`(12345678-12234-1234-1234-123456789012),
    `version`(1.1),
    `endpoint`("ncacn_ip_tcp:[1234]")
  ]
  ```
IDL File Structure. IDL Definition Body

- details all the procedures available for clients

```c
interface myinterface
{
    ... definitions ....
}
```

- example

```c
interface myinterface
{
    int  RunCommand([in] int command,
                    [in, string] unsigned char * args,
                    [out,string] unsigned char * res);
}
```
IDL File Structure. Compiler Features

- attributes like "size_is", "length_is", "range"
- compiler option "/robust"
- example

```c
interface myinterface
{
    int SendCommand([in, range(0,16)] int msg_id,
                    [in, range(0,1023)] int msg_len,
                    [in,length_is(msg_len)] unsigned char * msg);
}
```
Outline

1. Windows IPC Security
   - The Redirector
   - Impersonation

2. Window Messaging

3. Pipes

4. Mailslots

5. Remote Procedure Calls
   - RPC Connections
   - RPC Transports
   - Microsoft Interface Definition Language (MIDL)
   - Application Configuration Files (ACF)
   - RPC Servers
   - Impersonation in RPC
   - Context Handles and State
   - Threading in RPC
Application Configuration Files

- each interface has ACFs
- describe attributes that are local to the client or server and affect certain behavior
- have the same syntax like IDL, except the attributes they specify do not alter the interface definition
Outline

1. Windows IPC Security
   - The Redirector
   - Impersonation

2. Window Messaging

3. Pipes

4. Mailslots

5. Remote Procedure Calls
   - RPC Connections
   - RPC Transports
   - Microsoft Interface Definition Language (MIDL)
   - Application Configuration Files (ACF)
   - RPC Servers
     - Impersonation in RPC
     - Context Handles and State
     - Threading in RPC
Registering Interfaces

- function `RpcServerRegisterIf()`
- function `RpcServerRegisterIfEx()`
- flags relevant to security
  - `RPC_IF_ALLOW_CALLBACKS_WITH_NO_AUTH`
  - `RPC_IF_ALLOW_LOCAL_ONLY`
- function `RpcServerRegisterIf2()`
- example 1

```
RpcServerRegisterIfEx(hSpec, NULL, NULL, 0, 20, NULL);
```

- example 2

```
RpcServerRegisterIfEx(hSpec, NULL, NULL, RPC_IF_ALLOW_LOCAL_ONLY, 20, MyCallback);
```
Binding to an Endpoint

- after registering an interface, the server needs to bind to endpoints so that clients can contact it
- register protocol sequences that the server should accept connections on
- functions `RcpServerUseProtseq()`, `RcpServerUseProtseqEx()`, and `RcpServerUseAllProtseqs()`
- register the endpoints for each protocol sequence
- the endpoint is protocol-specific information required for contacting the RPC server
function `RpcEnRegister()`

provides the endpoint mapper with the endpoints of an RPC interface
Listening for Requests

- function `RocServerListen()`
- indicates that the server is expecting requests and potentially exposed to malicious input
attack surface of an RPC server depends heavily on the level of authentication it requires

- RPC_C_AUTH_LEVEL_DEFAULT
- RPC_C_AUTH_LEVEL_NONE
- RPC_C_AUTH_LEVEL_CONNECT
- RPC_C_AUTH_LEVEL_CALL
- RPC_C_AUTH_LEVEL_PKT
- RPC_C_AUTH_LEVEL_PKT_PRIVACY
- RPC_C_AUTH_LEVEL_PKT_INTEGRITY
Authentication (cont.)

- each authentication service must be registered by calling `RpcServerRegisterAuthInfo()`
Authenticating Requests

- the server can provide a DACL for a binding
- two routines can be used in a security callback
  - `RpcBindingInqAuthClient()`
  - `RpcServerInqCallAttributes()`
Outline

1. Windows IPC Security
   - The Redirector
   - Impersonation

2. Window Messaging

3. Pipes

4. Mailslots

5. Remote Procedure Calls
   - RPC Connections
   - RPC Transports
   - Microsoft Interface Definition Language (MIDL)
   - Application Configuration Files (ACF)
   - RPC Servers
   - Impersonation in RPC
   - Context Handles and State
   - Threading in RPC
Impersonation in RPC

- RPC can impersonate clients as named pipes.
- It is the effective method for accessing secure objects safely in the calling user’s context.
- Functions `RpcImpersonateClinet()` and `RpcGetAuthorizationContextForClient()`.
- Clients can restrict server’s capability to impersonate them, by using `RpcBindingSetAuthInfoEx()`.
- Levels of impersonation:
  - `RPC_C_IMP_LEVEL_DEFAULT`
  - `RPC_C_IMP_LEVEL_ANONYMOUS`
Impersonation in RPC (cont.)

- RPC_C_IMP_LEVEL_IDENTIFY
- RPC_C_IMP_LEVEL_IMPERSONATE
- RPC_C_IMP_LEVEL_DELEGATE
Outline

1. Windows IPC Security
   - The Redirector
   - Impersonation

2. Window Messaging

3. Pipes

4. Mailslots

5. Remote Procedure Calls
   - RPC Connections
   - RPC Transports
   - Microsoft Interface Definition Language (MIDL)
   - Application Configuration Files (ACF)
   - RPC Servers
   - Impersonation in RPC
   - Context Handles and State
   - Threading in RPC
Context Handles and State

- RPC is stateless, but provides explicit mechanisms for maintaining stateless.
- Typical mechanism: context handle.
- A unique token a client can supply to a server, similar in function to a session ID stored in a HTTP cookie.
- A context handle could be exposed to malicious users in a variety of ways:
  - Sniffing the network transport.
  - Through the actions of a malicious client.
  - Another RPC interface might even reveal the context handle if strict context handles are not used.
Strict Context Handles

- RPC service normally accepts any valid context handle, regardless of the originating interface.
- Developers can prevent this by using strict context handles.
- Defined by "strict_context_handle" attribute.
- Is valid only for the originating interface.
Proprietary State Mechanisms

- look for the following vulnerabilities
- predictable sessions identifiers
- short session identifiers vulnerable to brute-force attacks
- discoverable session identifiers
- session identifiers that leak sensitive information
Outline

1. Windows IPC Security
   - The Redirector
   - Impersonation
2. Window Messaging
3. Pipes
4. Mailslots
5. Remote Procedure Calls
   - RPC Connections
   - RPC Transports
   - Microsoft Interface Definition Language (MIDL)
   - Application Configuration Files (ACF)
   - RPC Servers
   - Impersonation in RPC
   - Context Handles and State
   - Threading in RPC
Threading in RPC

- RPC subsystem services calls via a pool of worker threads
- an RPC call can occur on any thread
- an RPC call can be preempted at any time, even by another instance of the same call
- such behavior could lead to vulnerabilities when access to shared resources is not synchronized properly