Malware Analysis

- Calling Conventions
  - cdecl
    - Caller cleans up the stack
  - standard
    - Callee cleans up the stack
  - Fast call
- Tools
  - PE File
    - Static Analysis
      - PeStudio
      - strings
        - --encoding=l --> for unicode strings
        - Bintext -> windows UI tool
      - CFF Explorer
      - HxD
      - Packed or not?
        - Detect it Easy
        - Bytehist
          - This shows the byte-usage histogram of binaries
            High entropy means packed
      - Powershell
        - powershell_ise
      - peStr and peframe
        - Linux tools for PE
      - strdeob.pl
        - This can be used to construct stack strings which is a obfuscating technique
      - FLOSS
        - string decoding utility
        - examines the file and emulates functions that can decode strings to determine what the output will be
        - Capable of getting stack strings as well

- Behaviour Analysis
  - Process Explorer
  - Process Monitor
  - ProcDot
  - RegShot
  - WireShark
  - INetSim
    - Emulates HTTPS and many other network protocols
    - /var/lib/inetsim/service.log file shows connection details
    - FakeNet-NG -> for windows
  - Fiddler
  - fakedns
    - ApateDNS -> for windows
  - TcpLogView
  - reg_export
    - Sometimes registry values cannot be exported easily by regedit because of some defenses applied by malware author. this tool can be used for this
  - Scylla
    - Dump the unpacked code from memory
  - Running Shellcode
    - scdbg
      - Emulates the execution of shellcode
    - jmp2it
      - Actually runs the SC
        - jmp2it.exe <file> <shellcode offset> pause

          This command will pause the execution just before shellcode so that we can attach debugger to it for analysis
        - 'addhandle <filename>'

          Used when analysing shellcode that needs document to run some commands. See [Shellcode Things!]
    - shellcode2exe.py
      - (to convert shellcode into a executable)
- Code Analysis

- IDA Pro
  - Powerful Disassembler
  - Has a DB of code signatures for common software libraries called FLIRT
- Debuggers
  - x64Dbg, x32Dbg
    - "ScyllaHide" plugin has the capability to remove malware defences
    - Breakpoints -> SetBPX <functionName>
  - WinDbg
  - OllyDbg
- Radar2
  - Toolkit for Windows and Linux
- Hopper
  - Disassembler and decompiler that runs on OSX

- PDF Files
  - pdfid
  - pdf-parser
  - peepdf.py
  - base64dump.py
    - To extract and decode base64 sequence from PDFs
    - -e pu -> percent unicode %u
    - -e bu -> \u encoded
- Microsoft Office Files
  - Doc
    - olevba
    - oledump
      - -i -> prints the general info about the stream
    - pcodedmp.py
      - This is to extract VBA macro p-code embedded in office. see [Malicious Web and Doc files][Office Macros] for more info
    - OfficeMalScanner.exe
    - SSViewer (Structured Storage Viewer)
    - zipdump.py
      - Examine all the files present in an office sample

        - -y <yara rules file> -C <decoder options> <file>
        - -e <file name> -> extended information
    - RTF
        - rtfdump
            - Tags:
              c = children
              p = location in hex
              l = group's length
              h = hex character's numbers
              b = binary characters
              O = if embedded object present
              u = unexpected characters
                - rtfdump.py <file> -s <obj num> -H -c start:end -d >
        - rtfobj.py
    - excel
    - PPT
- Deobfuscating Scripts Using
    - Debuggers
        - FireBug
            - Cannot place breakpoints in the middle. So, we have to modify the script and add a line break then put bp.
            - This modification can cause issues while deobfuscation process if script has 'document.callee' method implemented
                - This method allows function to refer to it's own body
        - IE Debugger
            - place "debugger;" in the html file and then load. Before allowing blocked content hit 'F12' to open developer tools and then reload the page
            - Ability to set breakpoint in the middle of the line
    - Interpreters
        - SpiderMonkey (js) -> Firefox
          cscript -> IE
          v8 -> chrome
            - SM and v8 -> print to print input.
              with CScript -> WScript.Echo
- Misc
    - CapTipper
        - To extract embedded files in a PCAP file

- NetworkMiner
  - Same as CapTipper
- box-js
  - --download -> to download the files that malicious JS script attempts to download
- xor-kpa .py
  - automatically derive XOR key by examining plaintext and encrypted text
- XORSearch
  - When invoked with -w parameter, tool examines file for commonly seen shellcode patterns

- Malicious Web and Doc files
  - Javascript
    - Statements
      - document.createElement() -> allows scripts to generate elements
      - Approaches to execute newly deobf scripts
        - document.body.appendChild() -> direct the browser to incorporate additional objects in web page
        - document.parentNode.insertBefore()
        - document.write()
        - eval
      - escape() and unescape() -> URL encoding and decoding
      - console.log() -> for printing variables in the debugger's console window
    - In javascript, tuples are designed to hold only one value. the last one
  - Office Macros
    - Two office formats
      - Binary format -> OLE2 (Object Linking and Embedding)
      - XML based file format -> Office Open XML (OOXML)
        - This will ignore all the macros unless the name contains m like .docm, .xlsm etc
        - In this, if macro is present, it is embedded inside a binary file that follows OLE2. Everything is zipped in one file
    - Numerous streams are present one of which is "SRP stream".
      - It contains compiled version of VBA macros
      - Contents of SRP stream might contain cached copy of VBA macros present in earlier version of malicious document

- Microsoft office stores the macros in two forms
  - source code
    - This is analyzed by the tools like olevba, oledump etc
      - Hacker can remove this completely while keeping p-code intact to hide
  - p-code -> compiled bytecode form
- RTF Documents
  - RTF don't support Macros
  - They allow arbitrary files to be embedded as objects using OLE1
    - This embedded objects could be any kind of file
  - Structure
    - Control word -> starts with "\"
      - embedded objects -> \object
    - groups -> {...}
    - Ex:
      {\object ...{*\objdata
      019393
      930022
      .....
      }}
- Additional Document Files
- PDFs (Portable Document Format)
  - Layout
    - Header -> info about the version of PDF
    - Obj
      - Syntax
        - 1 0 obj
          Type: /page
          <<
          /AA /O 43 0 R
          >>
          endobj
      - Malicious PDFs use streams to store data.
        - ----
          stream
          encoded contents
          endstream

- xref table
- Trailer
  - vital details:
    - offset of xref table
    - number of objects
    - root object
    - metadata of the file
- Dictionary entries (items b/w << >> are called)
  - /AA -> Automatic Action
    /O -> Open
    R -> Refers
    /OpenAction -> Action to take after file is opened
    /F -> File
    /P -> Parameters
    /AcroForm or /XFA -> interactive forms
- Web

- APIs
  - Downloading from C&C
    - - InternetOpenA()
      - InternetConnectA()
      - HTTPOpenRequestA()
      - InternetQueryOptionA()
      - InternetSetOptionA()
      - HTTPSendRequestA()
      - InternetReadFile()
      - URLDownloadToFileA() -> urlmon.dll
  - RtlDecompressBuffer
    - Decompress contents of buffer using diff compression format
  - Loading Resources
    - - FindResourceA()
      - LoadResource()
      - LockResource()
      - SizeofResource()
- Injection and hooking (Covering tracks)
  - DLL injection
    Ten Process Injection Techniques: A T…

- - CreateToolhelp32Snapshot() -> get a listing of running process
  - OpenProcess()
  - GetProcAddress(GetModuleHandle())
  - VirtualAllocEx()
  - WriteProcessMemory()
  - CreateRemoteThread()
  - Process32FirstW() -> To determine which process to inject
  - Process32NextW()
- SetWindowHookEx() method
  - See [KeyLoggers tab]
- Process Hollowing
  - - CreateProcessA() -> SUSPENDED mode (arg CreationFlag value = 4)
    - NtUnmapViewOfSection()
    - VirtualAllocEx()
    - WriteProcessMemory()
    - GetThreadContext()
    - SetThreadContext()
    - ResumeThread()

- Hooking
  - Inline hooking
    Windows Inline Function Hooking
    - ReadProcessMemory() -> to read target function's contents
      WriteProcessMemory() -> to overwrite the contents of the function
      - Together these are potential indication of API hooking
      - Generally, attackers remove the beginning of function and replace it with jmp (0xE9) or push <> (0x68 ...), ret (0xC3) commands
    - Call <API> -->
      jmp <addr> (initial 5 bytes changed) -->
      malicious code -->
      initial inst. of <API> that was replaced, jmp <addr of API after initial 5 bytes>
      (This step is in Trampoline function) -->
      body of API
  - Call table hooks (IAT hooks)
  - GUI hooks

- Shellcode things!
  - GetEIP
    - CALL followed by POP instruction to get the EIP
  - To call any APIs, it needs to first get address of kernel32.dll

- FS stores address of Thread Information Block (TIB) (in 32-bit system, in 64-bit GS register is used for this). AT 0x30 offset, PEB is present.
- PEB stores all the info about that process, like what all DLLs are loaded and where

- Sometimes shellcode in document files store code within document.
  - To read those contents, it iterates through all file handles until it finds the one that points to this document.
  - GetFileSize() can be used to check that the correct file handle is got by shellcode

- Malware Defences
  - Anti-Debugging Techniques
    - IsDebuggerPresent()
    - CheckRemoteDebuggerPresent()
    - NtQueryInformationProcess()
      - ProcessDebugFlags
      - ProcessDebugPorts
    - NtSetInformationThread Debugger detaching
    - BeingDebugged bit in the PEB
      - MOV EAX, FS:[30h]
        MOV EAX, [EAX+2]
        TEST EAX, EAX
    - LookupPrivilegeValue() with argument SeDebugPrivilege
    - Program Speed
      - GetTickCount()
        - Gets the number of millisec elapsed after system started
      - GetLocalTime()
      - GetSystemTime()
      - NtQuerySystemTime
      - RDTSC
        Read Time-Stamp Counter
        - Assembly instruction
    - BlockInput()
      - API blocks keyboard and mouse input events from reaching application, hence no debugging
  - Anti-VM
    - Number of Processors present
      - PEB -> FS:[30]
        Offset 0x64 -> NumberOfProcessors

- Detect virtualization
  Talos Blog || Cisco Talos Intelligence …
  - presence of virtualized Hardware like drivers, GUID etc
  - Descriptor table register check
    - There us only 1 IDTR (Interrupt Descriptor Table Register), GDTR (Global Descriptor) and LDTR (Local) per processor. Since there are 2 OSes running, diff will be there
    - SIDT, SGDT and SLDT are the assembly instructions to get these IDTR, GDTR and LDTR values
  - User Interaction like keypress or mouse movement
    Don’t Click the Left Mouse Button: In…
  - Check for Internet connection
  - Process Attributes
    - Loaded modules, attached debuggers etc
- Misdirection Techniques
  - API hooking using PUSH, RET
    Check [Hooking][Inline hooking]
  - Malicious use of SEH
    - It's a chain of memory addresses of functions which define error handling
    - SEH structure comprise of two memory addresses.
      - pointer to error handling function
      - pointer to previously defines SEH record
    - - SEH implementation which uses stack are called "frame based EH". 32-bit uses this
      - 64-bit implements SEH using "table based" approach, in which compiler creates a table that describes EH
      Talos Blog || Cisco Talos Intelligence …
    - Ex:
      mov EAX, <address of shellcode or attacker's code>
      push EAX <- pointer to EH function
      push fs:[0] <- next EH handler
      mov fs:[0], ESP
      <Cause any exception>
  - TLS Callbacks
    - Code private to each thread which runs before the entry point.
- Anti-Disassembly
- Obfuscation techniques
  - Stack strings

- XOR encoding

- Packed Malwares
  - Packed malware usually changes the following
    - The entry point correspond to the beginning of unpacking code.
      The beginning of original code, when unpacked, is called Original entry point (OEP)
    - IAT is often missing or incomplete
  - Analyzing Packed Malwares
    - Dump the unpacked code using following tools from memory
      - Scylla
        - Have some limitations in terms of fixing OEP (original entry point)
      - OllyDumpEx plugin present in x64dbg to fix OEP and then dump
        - And then using Scylla plugin in x64dbg to fix IAT in the dumped binary
    - Debugging
      - Let the malware run so that it can unpack itself in memory
        Look for the memory region where it could have unpacked itself. e.g., look for memory region with 'E' flag
        Then we can look for import functions and all
  - Packers
    - UPX
    - PECompact
  - Unpacking approaches
    - Stack cleanup and reuse
    - WriteProcessMemory
    - RtlDecompressMemory
    - VirtualAlloc in shellcode

- Persistence Mechanisms
  - Modifying Registry keys
    - Run/RunOnce keys
    - Keys used in Winlogon process
      - Can hook malware to a partiicular Winlogon event ex., logon, logoff, startup, shutdown, lock screen
      - UserInit Key present at HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon

- Notify HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Notify
- Startup Keys
  - Placing files under startup directory. Start locations are provided by these keys
    - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders
    - HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
    - HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
    - HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders
- Services
  - HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services.
  - Malicious files can be loaded if a service fails to start. There is a recovery tab in service properties which can be changed to run a program on failure
- AppInit_DLLs
  - HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs
    - This shows which all DLLs are loaded along with user32.dll
- Image File Execution Options (IFEO)
  Image File Execution Options (IFEO) |…
- DLL Search order hijacking
  - HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\KnownDLLs --> Known DLLs registry key
    - If OS can't find DLL at these, then DLL search starts in following order
      - Directory from where application was launched
      - System Directory(C:\Windows\System32)
      - Windows Directory
      - Current Working Directory
      - Directories defined in the PATH variable.

- Malware Behaviour
  - Credential Stealing
    - KeyLoggers
      - Install a hook for keyboard related events

- SetWindowHookExA()
  - Often used to install hook procedures
  - only works with graphical applications
  - SetWindowsHookEx can be used to inject a DLL into another process. A 32-bit DLL cannot be injected into a 64-bit process, and a 64-bit DLL cannot be injected into a 32-bit process.
    Using SetWindowsHookEx for DLL In…
- Poll the state of each key
  - GetKeyState()
    - Determines if the key is currently pressed
  - GetAsyncKeyState()
    - if key is pressed currently or was pressed since previous call
- Hash Dumping
  - APIs used for this after injection into lsass.exe:
    - GetHash()
  - pwdump and Pash-the-Hash (PSH) toolkits are present]
- Downloaders and Launchers
  - UrlDownloadToFileA followed by WinExec()
- Backdoors
  - Provides remote access of victim machine

- Memory Forensics
  - Moved to different Mindmap
- Code Life Cycle
  - - Source Code
    - Compiler converts this into Object code
    - Linker links all the libraries and form Executable File
    - OS (Loader) loads this Exe file to assembly code