Win32 Virtual Memory - Process Structures

THREAD ENVIRONMENT BLOCK X64

TEB IS THE ONLY VIRTUAL PROCESS STRUCTURE EASILY OBTAINABLE WITHOUT USING WINDOWS API. IT'S VIRTUAL ADDRESS IS STORED IN GS REGISTER DURING RUNTIME, SO YOU CAN CONSIDER IT THE STARTING POINT TO EXPLORING USERSPACE PROCESS DATA

STRUCTURE START ON ADDRESS STORED IN GS REGISTER

0x00	ULONG	EXCEPTION_LIST	POINTER TO EXCEPTION LIST
0x08	ULONG	STACK_BASE	POINTER TO STACK START
0x10	ULONG	STACK_LIMIT	POINTER TO MIN STACK ADDRESS
0x18	ULONG	SUBSYSTEM_TIB	
0x20	ULONG	FIBER_DATA	
0x28	ULONG	ARBITRARY_USER_PTR	
0x30	ULONG	SELF	POINTER TO THIS STRUCTURE
0x38	ULONG	ENVIRONMENT_PTR	
0x40	ULONG	UNIQUE_PROCESS	
0x48	ULONG	UNIQUE_THREAD	
0x50	ULONG	ACTIVE_RPC_HANDLE	
0x58	ULONG	TLS_POINTER	
0x60	ULONG	PEB_POINTER	POINTER TO PROCESS ENVIRONMENT BLOCK.
...	...	...	

EXCEPTION HANDLER FRAME - CODE SNIPPETS THAT ARE BEING RUN WHEN EXCEPTION IS CATCHED BY THE CODE PROVIDED BY PROGRAMMER. CAN BE OVERWRITTEN WITH MALICIOUS CODE

0x00	ULONG	POINTER TO NEXT SEH HANDLER
0x08	ULONG	HANDLER CODE POINTER
0x00	ULONG	POINTER TO NEXT SEH HANDLER
0x08	ULONG	HANDLER CODE POINTER

```
EXCEPTION HANDLER CODE
CATCH ( ) {
    ...
}
```

OVERWRITING THE SEH HANDLERS WITH MALICIOUS CODE CAN BE ESPECIALLY DANGEROUS SINCE IT'S EXECUTION CANNOT BE EASILY DETECTED. CODE IS BEING RUN ONLY WHEN EXCEPTION IS RAISED. PRECISELY, THE SEH FRAME IS BEING PUSHED ONTO THE STACK. WHEN THE EXCEPTION TAKES PLACE, THE OVERRITEN "NEXT SEH" POINTER SHALL POINT TO MALICIOUS CODE. PROTECTION AGAINST OVERWRITING SEH HANDLERS MAY BE ENABLED WITH SPECIAL ADMX POLICY

IMAGE BASE ADDRESS

EXACT ADDRESS PF PE FILE MAPPED IN VIRTUAL MEMORY. REMEMBER THAT PE MAPPED IN MEMORY IS ALREADY RELOCATED, SO SECTIONS HAVE NEW OFFSETS AND SIZES. WHEN NAVIGATING THROUGH PE FILE, MAKE SURE TO CONVERT RVA'S TO 64-BIT AND ADD IT TO THE AFORMENTIONED BASE IMAGE ADDRESS.

TO FIND THE RVA'S, MAKE SHURE TO CHECK THE POSTER DOCUMENTING THE PE FILE STRUCTURE AND DETAILS.

PROCESS ENVIRONMENT BLOCK X64

MOST INTERESTING VIRTUAL PROCESS STRUCTURE. WITH PEB YOU CAN ACCESS THE LOADED PE FILE AS WELL AS DLLS, ALSO PROCESS PARAMETERS, CURRENT DIRECTORY, CMDLINES AND OTHER PROCESS PROPERTIES ARE GATHERED IN THIS STRUCTURE.

0x002	BOOL	IF TRUE, DEBUGGER IS PRESENT
0x010	ULONG	BASE IMAGE ADDRESS (PE FILE)
0x018	ULONG	WINDOWS LDR STRUCTURE ADDRESS
0x020	ULONG	PROCESS PARAMETERS
0x030	ULONG	PROCESS HEAP POINTER
0x0B8	DWORD	NUMBER OF PROCESSORS
0x0E8	DWORD	NUMBER OF HEAPS
0x0EC	DWORD	MAX NUMBER OF HEAPS
0x0f0	ULONG	PROCESS HEAPS POINTER
0x118	DWORD	OS MAJOR VERSION
0x11C	DWORD	OS MINOR VERSION
0x120	WORD	OS BUILD NUMBER
0x128	DWORD	IMAGE SUBSYSTEM TYPE
0x12C	DWORD	IMAGE SUBSYSTEM MAJOR VERSION
0x130	DWORD	IMAGE SUBSYSTEM MINOR VERSION
0x2C0	DWORD	SESSION ID
...	...	...

PROCESS PARAMETERS

0x020	ULONG	STANDARD INPUT	HANDLE TO STANDARD INPUT
0x028	ULONG	STANDARD OUTPUT	HANDLE TO STANDARD OUTPUT
0x030	ULONG	STANDARD ERROR	HANDLE TO STANDARD ERROR
0x038	ULONG	CURRENT DIRECTORY	PTR TO CURRENT DIRECTORY STRING
0x050	ULONG	DLLPATH	PATH TO IMAGE STRING
0x060	ULONG	IMAGE PATH NAME	IMAGE PATH NAME STRING
0x070	ULONG	COMMANDLINE	PTR TO COMMANDLINE STRING
0x080	ULONG	ENVIRONMENT	ENVIRONMENT VARS ETC.
0x060	ULONG	WINDOW TITLE	PTR TO WINDOW TITLE STRING
...	...	...	...

STRUCTURE STORES VARIOUS PROCESS AND PE FILE PARAMETERS. AFORMENTIONED POINTERS HANDLES THE ADDRESSES OF UNICODE STRING STRUCTURES. ALSO , SOME PROPERTIES E.G. STANDARD INPUT STORE HEXADECIMAL NUMBER OF HANDLE ID CONNECTED WITH REQUESTED RESOURCE.
THESE HESXADECIMAL NUMBERS CAN BE ASSIGNED TO REDIRECT ANY OF THESE PROPERTIES TO OPEN HANDLE E.G. FILE,SOCKET OR IPC.

LUCAS GERAS 2024 - USE AT YOUR OWN RISK :)

Win32 Virtual Memory - Process Structures

PROCESS ENVIRONMENT BLOCK X64

MOST INTERESTING VIRTUAL PROCESS STRUCTURE. WITH PEB YOU CAN ACCESS THE LOADED PE FILE AS WELL AS DLLS, ALSO PROCESS PARAMETERS, CURRENT DIRECTORY, CMDLINES AND OTHER PROCESS PROPERTIES ARE GATHERED IN THIS STRUCTURE.

0x002	BOOL	IF TRUE, DEBUGGER IS PRESENT
0x010	ULONG	BASE IMAGE ADDRESS (PE FILE)
0x018	ULONG	WINDOWS LDR STRUCTURE ADDRESS
0x020	ULONG	PROCESS PARAMETERS
0x030	ULONG	PROCESS HEAP POINTER
0x0B8	DWORD	NUMBER OF PROCESSORS
0x0E8	DWORD	NUMBER OF HEAPS
0x0EC	DWORD	MAX NUMBER OF HEAPS
0x0f0	ULONG	PROCESS HEAPS POINTER
0x118	DWORD	OS MAJOR VERSION
0x11C	DWORD	OS MINOR VERSION
0x120	WORD	OS BUILD NUMBER
0x128	DWORD	IMAGE SUBSYSTEM TYPE
0x12C	DWORD	IMAGE SUBSYSTEM MAJOR VERSION
0x130	DWORD	IMAGE SUBSYSTEM MINOR VERSION
0x2C0	DWORD	SESSION ID
...	...	...

LDR (PE LOADER STRUCT)

PROVIDES LINKED LIST OF LOADED PE FILES IN VARIOUS ORDERS. THE LIST CONTAINS ACTUAL PE OF THE EXECUTABLE AS WELL AS DLL'S LOADED INTO MEMORY. MOSTLY, WHEN USING THIS STRUCTURE, INLOADORDERMODULELIST IS USED TO OBTAIN NTDLL FILE AND USE NATIVE WINDOWS API OR SYSCALLS NUMBERS.

0x010	ULONG	INLOADORDERMODULELIST
0x020	ULONG	INMEMORYORDERMODULELIST
0x030	ULONG	ININITIALIZATIONORDERMODULELIST

LIST ENTRY

UNIVERAL WINDOWS STRUCTURE, IMPLEMENTATION OF BIDIRECTIONAL QUEUE.
OFTEN OVERLAPS WITH OTHER STRUCTURES, GIVING POSSIBILITY TO EASILY ITERATE OVER VARIOUS STRUCTURES IN VIRTUAL MEMORY

0x00	ULONG	FORWARD LINK	POINTS TO THE NEXT ELEMENT
0x08	ULONG	BACKWARD LINK	POINTS TO PREVIOUS ELEMENT

OR

LDR DATA TABLE ENTRY

0X030	ULONG	DLLBASE	DLL ADDRESS
0X038	ULONG	ENTRYPOINT	ENTRY POINT ADDRESS
0X040	ULONG	SIZEOFIMAGE	SIZE OF PE FILE
0X048	ULONG	FULLDLLNAME	PATH AND NAME OF DLL
0X058	ULONG	BASEDLLNAME	NAME OF DLL

DLL PE FILE

USAGE OF VIRTUAL PROCESS STRUCTURES IN MALWARE DEV

MALWARE DEVELOPMENT PROCESS INVOLVES VARIOUS TECHNIQUES OF ACCESSING DATA NAD CODE, AND ALSO EVADING DETECTION VIA EDITING VARIOUS PROPERTIES OF THE WINDOWS PROCESS. YOU CAN DO THIS MODYFING VIRTUAL MEMORY STRUCTURES MENTIONED.

ESPECIALLY USING FUNCTIONS NOT PRESENT IN PE FILE'S IMPORT TABLE IS ONE OF THE BEST EFFORTS YOU CAN MAKE TO EVADE DBEING DETECTED BY EXAMINATOR.
OTHER THIG IS, WHEN DEVELOPING SHELCODES, YOU CAN'T RELY ON IMPORTS, SINCE THE CODE INJECTED IS POSITION INDEPENDENT AND HAS NO CONNECTION WITH VIRTUAL PROCESS STRUCTURES.

FIRST STEP ALL OF THESE TECHNIQUES IS ACCESSING THE THREAD INFORMATION BLOCK, SINCE IT'S ADDRESS IS ALWAYS AVAILABLE FROM THREAD CONTEXT. ADDRESS CAN BE OBTAINED THROUGH READING CONTENTS OF GS REGISTER (IN 64-BIT WINDOWS VERSION) OR FS REGISTER (32-BIT).

AFTER OBTAINING THE TEB FROM REGISTER YOU ALREADY HAVE THE KEY TO ACCESS THE VIRTUAL MEMORY. READING VALUE AT IT'S OFFSET 0x60, YOU CAN ACCESS ALSO PROCESS ENVIRONMENT BLOCK. MODYFYING STRINGS IN PROCESS_PARAMTERS STRUCTURE, YOU CAN MASQUERADE THE PROCESS UNDER ANOTHER NAME, CURRENT DIRECTORY, COMMANDLINE OR WINDOW NAME

ALSO, WHEN ITERATING THROUGH LDR STRUCTURE, YOU CAN OBTAIN ANY FUNCTION FROM LOADED DLL'S AND COPY IT'S ADDRESS. WITH LOADED FUNCTION ADDRESS YOU CAN EASILY RUN FUNCTION FROM POINTER, WITHOUT IT'S PRESENCE IN MALICIOUS PE FILE IMPORT TABLE.

AIN CASE OF USING THE NATIVE API FUNCTIONS IMPORTED FROM NTDLL.DLL FILE, YOU HAVE TO READ THE FUNCTIONS BODY AND FIND THE SYSCALL NUMBER, SO YOU CAN CALL IT AFTER PUSHING THE ARGUMENTS ON THE STACK. THIS TECHNIQUE WILL LET YOU ALSO HIDE FROM API MONITORING INCLUDING VARIOUS EDR SYSTEMS AND TOOLS LIKE APIMONITOR OR PROCMON.

USAGE OF THIS TECHNQUES CAN EASILY BY DONE BY USING ASSEMBLY LANGUAGE AND CRAFTING SHELLCODE. ALSO IN C OR RUST YOU CAN ACCESS THE GS AND FS REGISTERS VALUES USING IINLINE ASSEMBLY.

IN CLANG, THE EASIEST WAY TO MAP THE STRUCTURES IS TO USE WINDOWS.H HEADER FILE, CONTAINING THE PROTOTYPES OF ALL THE STRUCTURES MENTIONED ON THE POSTER.

MAKE SURE TO FAMILIARISE WITH CORE CONCEPTS LIKE POINTERS IN 64-BIT MEMORY AND RVA IN WINDOWS PE FILES IN ORDER TO USE THE INFORMATION INCLUDED IN DEVELOPMENT.

LUCAS GERAS 2024 - USE AT YOUR OWN RISK :)

Win32 Virtual Memory - Process Structures

ACCESSING VIRTUAL MEMORY STRUCTURES

YOU CAN ACCESS VIRTUAL MEMORY STRUCTURES USING PREDEFINED WINAPI FUNCTIONS, BUT I RECOMMEND ACCESSING IT USING INLINE ASSEMMBLY:

NOTE THAT CODE RETURNS THE POINTER TO PEB STRUCTURE BOTH FOR 32-BIT AND 64-BIT ARCHITECTURE - CODE HANDLES BOTH. THANKS TO USING PREPROCESSOR COMMANDS:

FOR 32-BIT WINDOWS, THE PEB POINTER IS STORED IN FS REGISTER WITH 0X30 OFFSET:

FOR 64-BIT WINDOWS THE PEB POINTER IS STORED IN GS REGISTER WITH OFFSET 0X60:

FUNCTION RETURNS THE POINTER TO THE VIRTUAL MEMORY. NOTE THAT FUNCTION RETURNS *VOID TYPE, WHICH MEANS POINTER TO THE ELEMENT WITHOUT TYPE.

NOTE, THAT ASSEMBLY IS WRITTEN USING INTEL (NOT AT&T) ASSEMBLY SYNTAX. TO USE THIS SYNTAX WITH GCC COMPILER. IN ORDER TO USE IT, USE THE MENTIONED COMPILATION AGUMENTS:

```
gcc  -masm=intel .\source_file.c -o .\target_executable.exe
```

```
void *GetPEB(void){
  register void *pPEB;
  #if defined(   x86_64   ) || defined(   amd64   )
    __asm__(
      "mov %0, gs:[0x60]"
      :"=r"(pPEB)
    );
  return pPEB;
  #elif defined(__i386__)
    __asm__(
      "movq %0, fs:[0x30]"
      :"=r"(pPEB)
    );
  return pPEB;
  #endif
}
```

MODIFYING DATA IN VIRTUAL MEMORY STRUCTURES

IN ORDER TO MODIFY THE DATA STORED IN STRUCTURES YOU HAVE TO GAIN ACCESS FIRST, USING OPENPROCESS FUNCTION. FIRST, IDENTIFY THE PID OF THE CURRENT PROCESS USING CETCURRENTPROCESSID FUNCTION. NOTE ONE OF THE ARGUMENTS, SPECIFYING THE ACCESS MODE: "PROCESS_ALL_ACCESS"

```
HANDLE pid = GetCurrentProcessId();
HANDLE h = OpenProcess(PROCESS_ALL_ACCESS,FALSE,pid);
```

THEN, FIND THE DATA ADDRESS YOU WANT TO MODIFY - IN THIS CASE, WE'LL BY MODYFING IMAGE BASE PATH PARAMETERS FROM PROCESS_PARAMETERS MEMBER OF PEB:

```
void* peb_address = GetPEB();
void* process_parameters = *(unsigned long long*)(peb_address + 0x20);
void* image_path_string_ptr = *(unsigned long long*)(process_parameters + 0x68);
```

CREATE THE STIRING YOU WANT TO OVERWRITE THE DATA WITH. NOTE THE STRING IS 2-BYTE PER CHARACTER::

```
wchar_t new_string[] = L"C:\\Windows\\System32\\notepad.exe";
```

PREPARE THE MEMORY TO OVERWRITE, BY ALLOWING ACCESS TO WRITE IT TO THE PROCESS::

```
VirtualProtect((LPVOID)image_path_string_ptr, sizeof(new_string), PAGE_EXECUTE_READWRITE, &Old);
```

FINALLY, WRITE THE DATA WITH USING WRITEPROCESSMEMORY FUNCTION:

```
WriteProcessMemory(h,image_path_string_ptr,&new_string,sizeof(new_string),NULL);
```

CHECK IF OVERWRITING IS SUCCEDDED, BY CHECKING IF GETLASTERROR RETURNS 0;:

```
int errorcode = GetLastError();
printf("Error:%x",errorcode);
```