CTF walkthrough

Challenge 1 - DebuggerThreat

Introduction

This is the first challenge of Minerva's Most Evasive CTF of 2022. This stage is mostly deals with basic anti-debugging techniques. Malware usually uses anti-Debugging techniques to ensure that it is not running under a debugger If it finds that it is, it might change its behavior accordingly. In most cases, the Anti-Debugging process will slow down the process of reverse engineering but will not prevent it.

Evasion Techniques

1. *IsDebuggerPresent – API call*
2. *CPUID Anti-VM*
3. *IsDebuggerPresent – assembly implementation*
4. *IsDebuggerPresent – API call*
5. *Hidden thread*
6. *Parent process window name*

IsDebuggerPresent – API call

The IsDebuggerPresent function Determines whether the calling process is being debugged by a user-mode debugger such as IDA, x64dbg, etc. Generally, the function only checks the 'BeingDebugged' flag of the Process Environment Block (PEB):

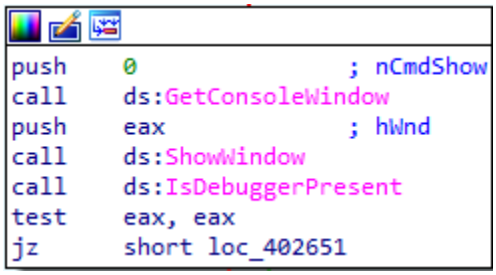

Figure 1 - IsDebuggerPresent API call

Bypass:

You can bypass this check by patching the return value in EAX register (set EAX register to zero) or patching the PEB flag (change the PEB BeingDebugged flag from 1 to 0).

CPUID Anti-VM

CPUID is an instruction-level detection method, and these kinds of methods are mostly hard to detect. This instruction is executed with EAX=1 as input, the return value describes the processor's features. The 31st bit of ECX on a physical machine will be equal to 0. On a guest VM it will equal 1.

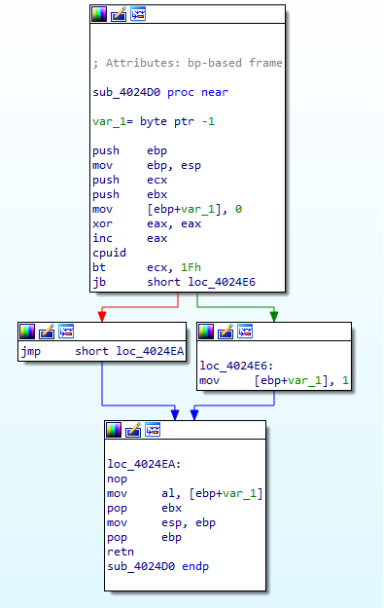

Figure 2 - CPUID Anti-VM

Bypass:

This check is performed in a separate function that returns 1 if running in VM and 0 if running on a physical machine. Patch the function return value in EAX register to bypass this technique.

IsDebuggerPresent – assembly implementation

An assembly implementation of IsDebuggerPresent API call.

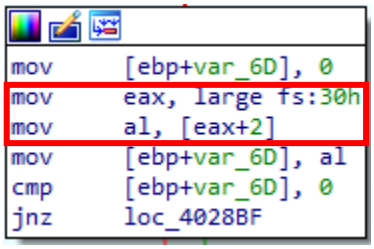

Figure 3 - IsDebuggerPresent - assembly implementation

The AL register will hold a BeingDebbuged flag value.

Bypass:

The bypass is the same as the regular IsDebuggerPresent check.

Hidden Thread

Threat actors might hide a thread from the debugger by using the NtSetInformationThread function with the undocumented THREAD_INFORMATION_CLASS::ThreadHideFromDebugger(0x11) value. After the thread is hidden from the debugger, it will continue running but the debugger won't receive any events related to it. This thread can then be used for whatever the threat actor desires.

Figure 4 - Hidden Thread Creation

Bypass:

To bypass this technique, patch the "11h" to "0", before NtSetInformationThread is called. This will create a regular thread; you can put a breakpoint in the newly created thread and continue in execution flow.

Parent process window name

Some malwares might check their parent process name. Usually, user processes run under explorer.exe. So a threat actor might check the parent process name and change the execution flow/exit if the parent process name is equal to one of the debugger names (IDA for example) or just different from what is expected:

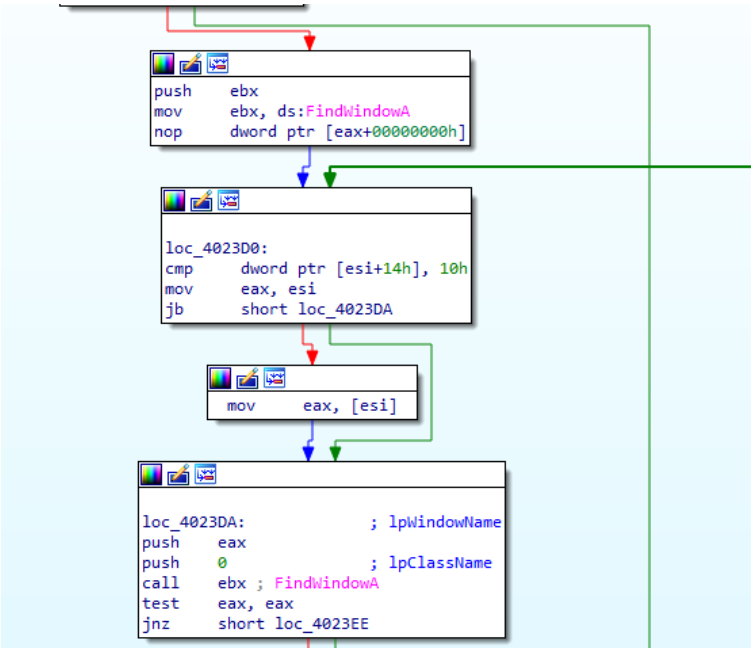

Figure 5 - Process Window Name check

Bypass:

When FindWindowA returns a non-zero value, you can patch it to 0, the search will run until all predefined debugger names are checked. I If at the end of the list the process is not exited, the thread will finish its work and return to the main thread which will decrypt the flag.

Flag: {minerva_labs_ctf:c2hv-dWxk-IHdh-a2Ug-dXAg-ZWFybHk}

Challenge 2 - OpenVault

Introduction

This stage is mostly about timing! When a process is traced in a debugger, there is a huge delay between instructions and execution. The “native” delay between some parts of code can be measured and compared with the actual delay using several approaches.

Evasion Techniques

1. *CPUID Anti-VM*
2. *VM port Anti-VM*
3. *QueryPerformanceCounter time Anti-debugging*
4. *NtGlobalFlag Anti-Debugging*
5. *QueryPerformanceCounter time Anti-debugging*
6. *IsDebuggerPresent – assembly implementation*
7. *QueryPerformanceCounter time Anti-debugging*

CPUID Anti-VM

CPUID is an instruction-level detection method, which is mostly hard to detect. This instruction is executed with EAX=1 as input, and the return value describes the processor’s features. The 31st bit of ECX on a physical machine will be equal to 0. On a guest VM it will equal 1.

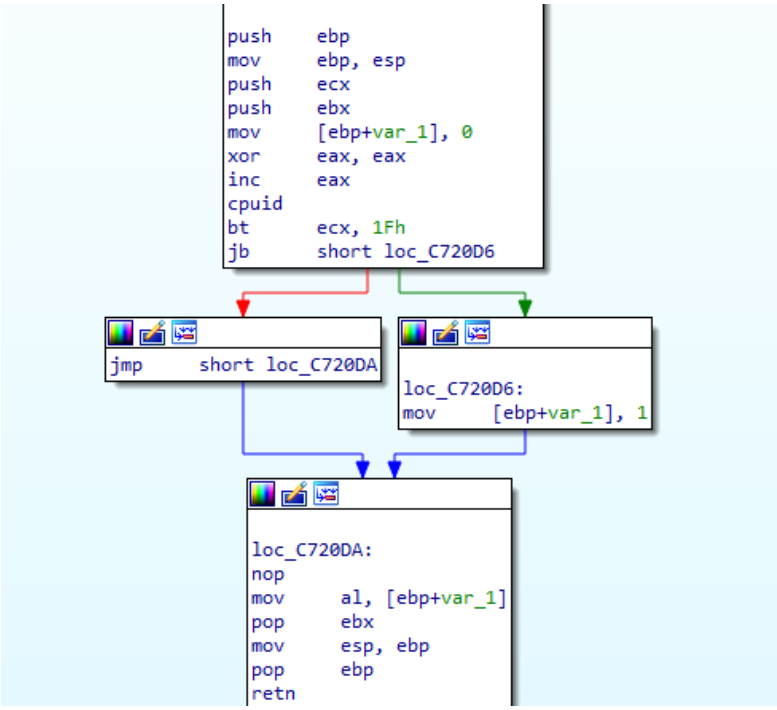

Figure 6 - CPUID Anti-VM

Bypass:

This check is performed in a separate function that returns 1 if running in a VM and 0 if running on a physical machine. Patch the function return value in eax register to bypass this technique.

VM Port Anti-VM

VMware uses the I/O port `0x5658` ("VX" in ASCII) to communicate with the virtual machine. Malware might detect the presence of that port as an Anti-VM check.

```
mov     [ebp+ms_exc.registration.TryLevel],
mov     eax, 564D5868h
mov     edx, 5658h
in      eax, dx
cmp     ebx, 564D5868h
```

Figure 7 - VM port Anti - VM

Bypass:

This check is performed in a separate function that returns 1 if running in VMware and 0 if not. Patch the function return value in the eax register to bypass this technique.

QueryPerformanceCounter time Anti-debugging

As we mentioned before, when a process is traced in a debugger, there is a delay between instructions and execution. The delay between some parts of code can be measured and compared with the actual delay. One of the ways to check for time delay is by using the QueryPerformanceCounter function, which, according to Microsoft documentation "Retrieves the current value of the performance counter, which is a high resolution (<1us) time stamp that can be used for time-interval measurements."

Figure 8 - QueryPerformanceCounter time Anti-Debugging

Bypass:

As in this check there is a comparison of the ECX register. We can change the value of ECX to be below or equal to the value it is compared to.

NtGlobalFlag Anti-Debugging

The NtGlobalFlag field of the Process Environment Block (0x68 offset on 32-Bit and 0xBC on 64-bit Windows) is 0 by default. **Attaching** a debugger doesn't change the value of NtGlobalFlag. However, if the process was **created** by a debugger, the NtGlobalFlag field will be set to 0x70.

```
mov     eax, large fs:30h
mov     al, [eax+68h]
and     al, 70h
cmp     al, 70h ; 'p'
```

Figure 9 - NtGlobalFlag value check

Bypass:

The easiest way to bypass this check is to patch the NtGlobalFlag field of the Process Environment Block (from 70h to 0).

IsDebuggerPresent – assembly implementation

The IsDebuggerPresent function Determines whether the calling process is being debugged by a user-mode debugger such as IDA, x64dbg, etc. Generally, the function only checks the BeingDebugged flag of the Process Environment Block (PEB).

In this challenge we used a direct assembly implementation of the IsDebuggerPresent API call.

```
mov     eax, large fs:30h
mov     al, [eax+2]
mov     [ebp-2F5h], al
mov     al, [ebp-2F5h]
mov     [ebp-2F5h], al
```

Figure 10 - IsDebuggerPresent assembly implementation

Bypass:

The best way to bypass this check is to patch the PEB flag (change the PEB BeingDebugged flag from 1 to 0).

Flag: {minerva_labs_ctf:SGFwc-HkgdG-8gb3B-lbiBt-eSBwa-G9uZQ}

Challenge 3 - ICanSeeNothing

Introduction

ICanSeeNothing hides the important part from the researcher inside a shellcode, which is injected into another process. The injection is implemented inside a hidden thread. Hiding threads from the debugger is a common evasion technique used by threat actors to execute malicious activity and prevent (or at least delay) the analysis.

Evasion Techniques

1. *GetLocalTime timing check*
2. *IsDebuggerPresent – API call*
3. *Hidden thread*
4. *Shellcode*

GetLocalTime timing check

When a process is traced in a debugger, there is a delay between instructions and execution. The delay between some parts of code can be measured and compared to the actual delay. One of the ways to check for time delay is by using the GetLocalTime function which retrieves the current local date and time.

```
lea     eax, [ebp+SystemTime]
push    eax             ; lpSystemTime
call    ds:GetLocalTime
```

Figure 11 - part of GetLocalTime Anti-Debugging function

Bypass:

This check is implemented in a separate function which returns 1 if a debugger is detected, or 0 if not. The best way is to change the returned value by this function to 0.

IsDebuggerPresent – API call

The IsDebuggerPresent function Determines whether the calling process is being debugged by a user-mode debugger such as IDA, x64dbg, etc. Generally, the function only checks the BeingDebugged flag of the Process Environment Block (PEB):

```
xor     eax, ebp
mov     [ebp+var_4], eax
call    ds:IsDebuggerPresent
test    eax, eax
jz      short loc_402CAB
```

Figure 12 - IsDebuggerPresent API call

Bypass:

You can bypass this check by patching the return value in EAX register (set eax register to zero) or by patching the PEB flag (change the PEB BeingDebugged flag from 1 to 0).

Hidden Thread

Threat actors might hide a thread from the debugger by using the NtSetInformationThread function with the undocumented THREAD_INFORMATION_CLASS::ThreadHideFromDebugger(0x11) value. After the thread is hidden from the debugger, it will continue running, but the debugger won't receive events related to this thread. This thread could then be used for whatever the threat actor desires.

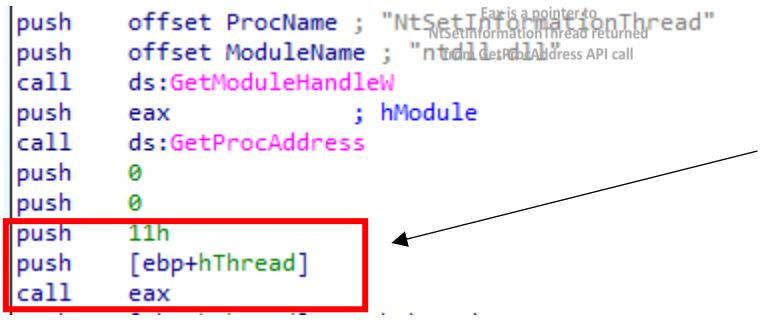

Figure 13 - Hidden Thread Creation

Bypass:

To bypass this technique, patch the "11h" to "0", before NtSetInformationThread is called. This will create a regular thread; you can set a breakpoint in the newly created thread and continue in execution flow.

Shellcode

The shellcode is injected into the parent process. The beginning of the shellcode is passed in 'lpBaseAddress' to WriteProcessMemory:

```
push    edx             ; lpBuffer
mov     eax, [ebp+lpBaseAddress]
push    eax             ; lpBaseAddress
mov     ecx, [ebp+hProcess]
push    ecx             ; hProcess
call    ds:WriteProcessMemory
```

Figure 14 - Injection

Bypass:

As the flag is passed in shellcode, we can dump it and compile it. There are two ways to dump the shellcode in this challenge. The first one is to dump it from the debugged process memory. The second, is to dump it after it is written to a target process. After the dump you can compile it using this small piece of code:

compile the shellcode.txt

Flag: {minerva_labs_ctf:VHJ5-IHRv-IGNh-dGNo-IG1l-IG5vdw}

Challenge 4 - What is our Name

This challenge is the middle stage of our Evasive CTF! It combines some evasion techniques that we already covered in previous challenges as well as new ones.

Evasion Techniques

1. *QueryPerformanceCounter time Anti-Debugging*
2. *PC Friendly Name Anti-VM*
3. *MAC Address Anti-VM*
4. *PC Friendly Name Anti-VM*
5. *DbgPrint()*
6. *Trap Flag*
7. *Int 3*
8. *PC Friendly Name Anti-VM*

QueryPerformanceCounter time Anti-debugging

As we mentioned before, when a process is traced in a debugger, there is a delay between instructions and execution. The delay between some parts of code can be measured and compared with the actual delay. One of the ways to check for time delay is by using QueryPerformanceCounter function, which by Microsoft documentation “Retrieves the current value of the performance counter, which is a high resolution (<1us) time stamp that can be used for time-interval measurements.”

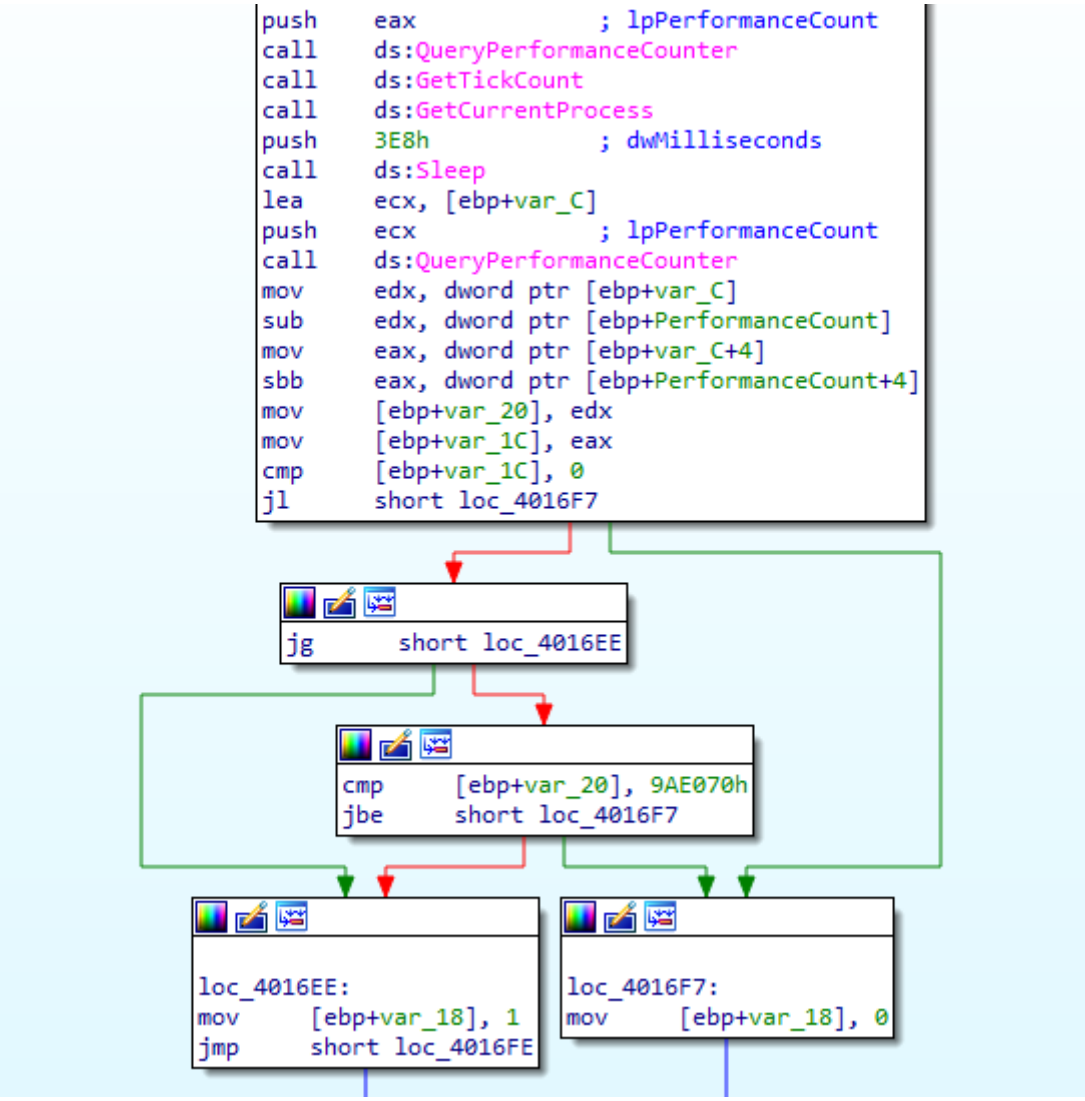

Figure 15 - QueryPerformanceCounter time Anti-Debugging

Bypass:

As in this check there is a comparison of a value in memory, we can change the value to be below or equal to the one it is compared to. Another way is to patch the value set in memory (from 1 to zero) or just change the returned value by this function to 0.

PC Friendly Name Anti-VM

Friendly names are assigned to complement unique identifiers that are comprised of numeric or alphanumeric code by default. The malware might retrieve a friendly name of all devices present on the current system by using SetupDiGetClassDevs, SetupDiEnumDeviceInfo and SetupDiGetDeviceRegistryProperty and checking for the occurrence of the strings VMWARE, VBOX, VIRTUAL HD and QEMU within the name:

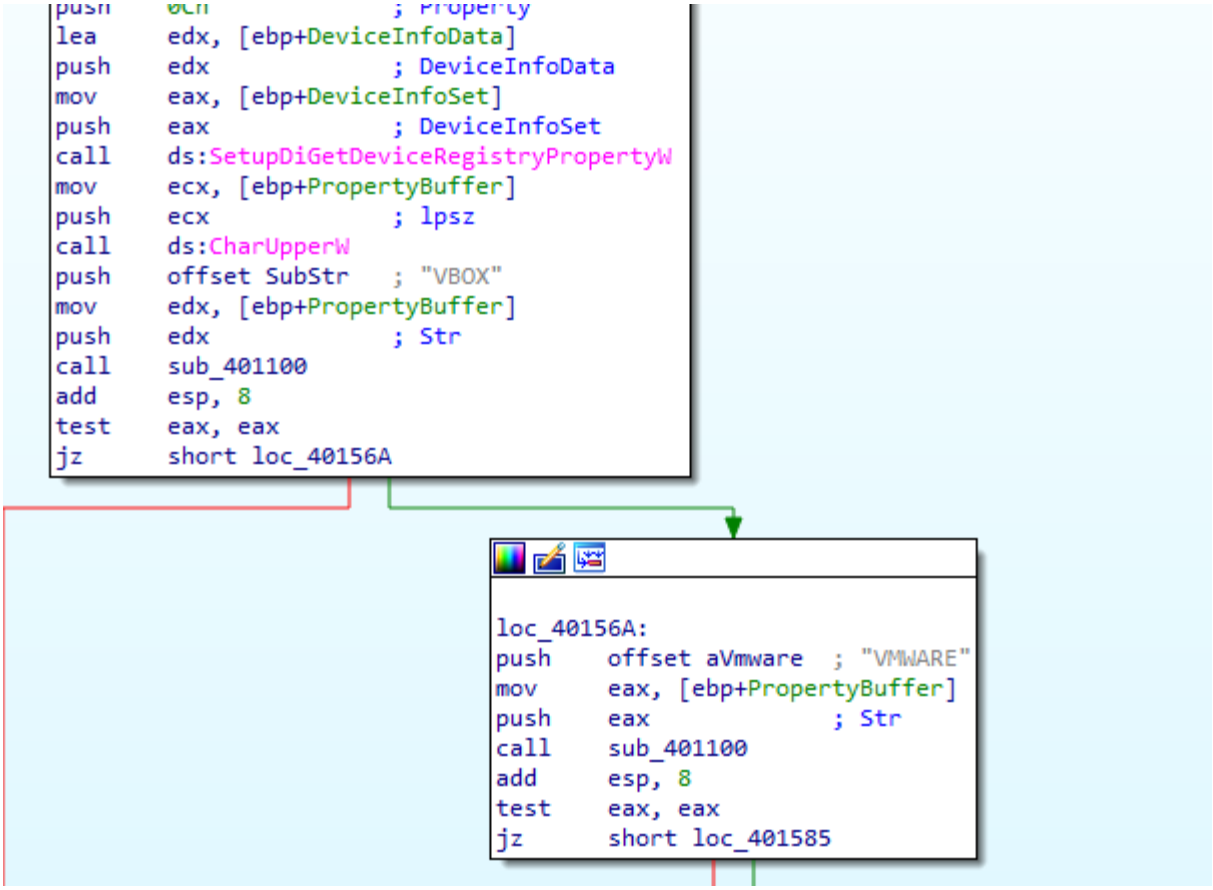

Figure 16 - part of Friendly Name check function

Bypass:

There is a function that checks friendly name and returns 1 if the name contains the required string (such as “VMWARE”) and 0 if the string was not found in friendly name. We might easily change the returned value from this function by changing the EAX register’s value to 0.

MAC Address Anti-VM

Vendors of different virtual environments hard code some values. MAC address is one of them. The malware might look for these values to detect a VM environment. The MAC address check might be implemented by using GetAdaptersInfo function which “retrieves adapter information for the local computer.” Usually, it is enough to check the beginning of the MAC Address do detect a VM:

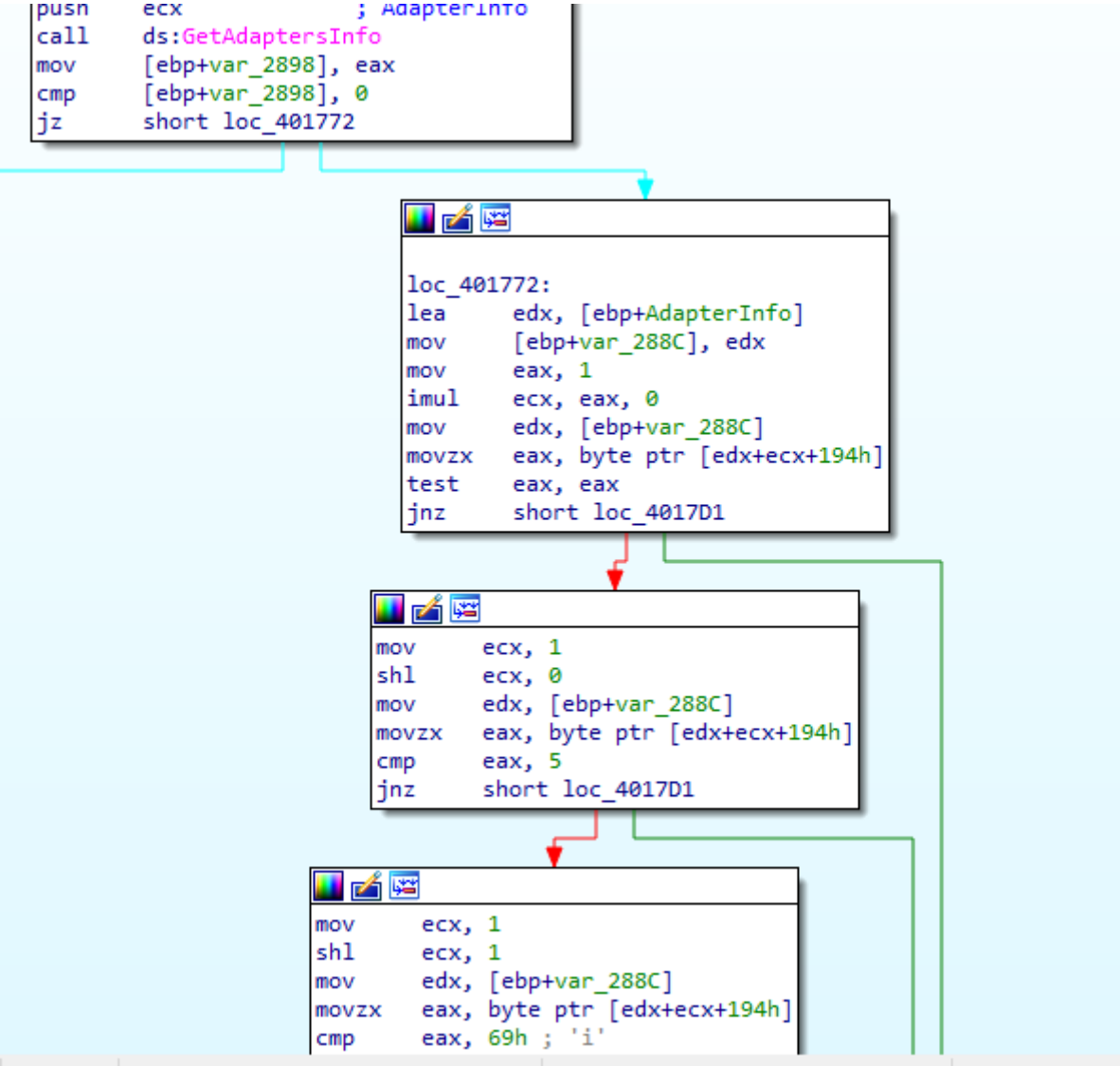

Figure 17 - part of MAC Address check function

Bypass:

There are two ways to bypass this check, the first is by changing MAC Address parts placed in the EAX register so it will not match the known VM MAC addresses. The second is to change the function return value (EAX register) to 0.

DbgPrint()

Some debug functions cause the exception DBG_PRINTEXCEPTION_C (0x40010006) and DbgPrint() is one of them. If a program is executed with an attached debugger, then the debugger will handle this exception. But if no debugger is present and an exception handler is registered, this exception will be caught by the exception handler.

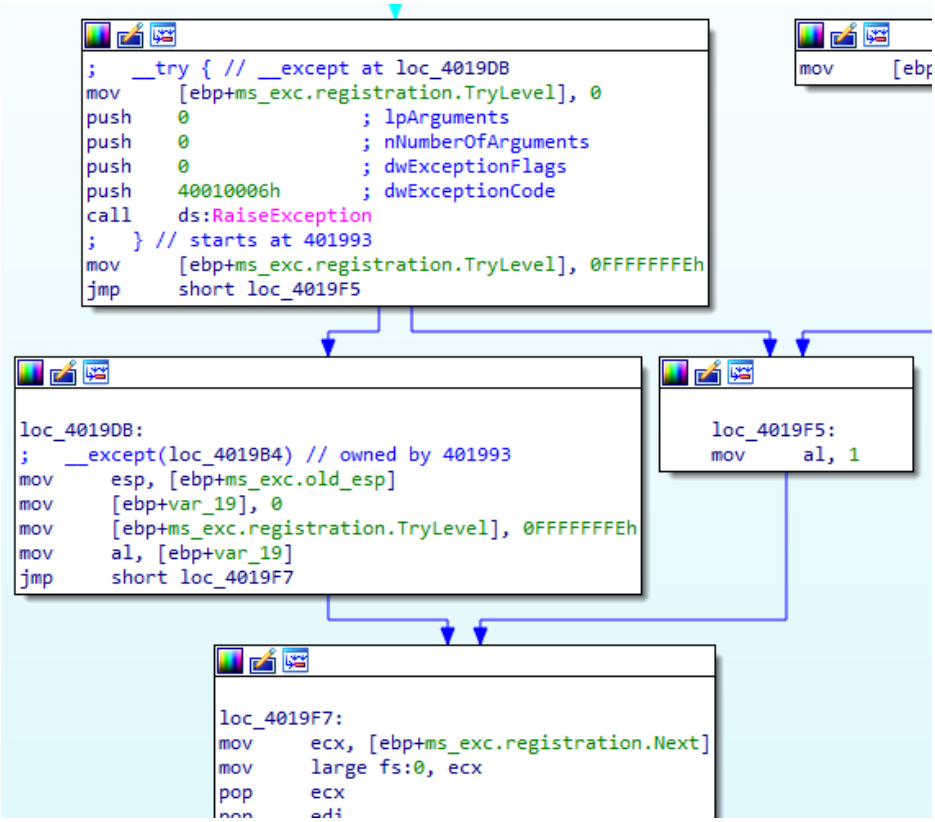

Figure 18 - DgbPrint Anti-Debugging

Bypass:

This check is implemented in a separate function which returns 1 if debugger is detected, or 0 if not. The best way is to change the returned value by this function to 0.

Trap Flag and int3 Anti-Debugging

- There exist several Flag Registers exists, one of them is a Trap Flag. When the Trap Flag is set, the single step exception is raised. However, if we traced the code, the Trap Flag will be cleared by the debugger, so we won't see the exception.
- INT3 is an assembly instruction which is used as a software breakpoint. Without a debugger present, after getting to the INT3 instruction, the exception EXCEPTION_BREAKPOINT (0x80000003) is generated, and an exception handler will be called. If the debugger is present, control won't be given to the exception handler.

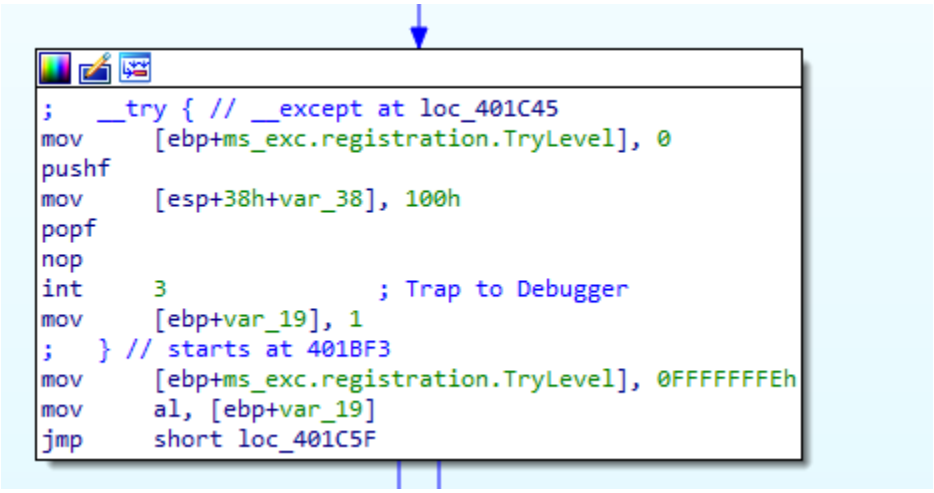

Figure 19 - Trap Flag check and INT3 Anti-Debugging

Bypass:

There are two bypasses needed:

1. For INT3 – we can bypass by patching the code and instead of INT3 (0xCC) placing the NOP instruction(0x90), or we can run up until the INT3 instruction and change the EIP register so it will execute the following instruction instead of INT3.
2. For Trap Flag – the easiest way is to run up until the JMP instruction after the AL register is set and change the AL register value to 0.

Flag: {minerva_labs_ctf:WW91-IGZv-dW5k-IG15-IE5h-bWU}

Challenge 5 - Who are you, John Doe

Introduction

Evasion Techniques

1. *Long sleep Anti-Sandbox*
2. *Debugger Attached Anti-Debugger*
3. *System temperature Anti-VM*
4. *Physical memory Anti-VM*
5. *MAC Address Anti-VM*
6. *Packer*

Long sleep Anti-Sandbox

Since they have limited resources, sandboxes typically analyze the execution of files for a limited time period, after which the analysis times out to free resources and move to the next file. Threat actors might abuse this and perform a long sleep at the beginning of execution. This way, the sandbox does not reveal the malicious file's real activity.

```
public static void sleep()
{
    Thread.Sleep(60000);
}
```

Figure 20 - 1 min sleep

Bypass:

When debugging the file, no bypass (only patience) is required. One might also patch the number of milliseconds the thread would sleep.

Debugger Attached Anti-Debugger

Debugger.IsAttached Property Gets a value that indicates whether a debugger is attached to the process.

```
private static char Is_debugger_present()
{
    bool isAttached = Debugger.IsAttached;
    char result;
    if (isAttached)
    {
        result = 'y';
    }
    else
    {
        result = 'n';
    }
    return result;
}
```

Figure 21 - Is debugger Attached check

Bypass:

The fastest way to bypass this check is to patch the ‘result’ value to ‘n’.

System temperature Anti-VM

This check is made through a Windows Management Instrumentation (WMI) request to pull the current temperature of the target hardware. Such queries can return the CPU temperature of up to seven thermal zones, as well as processor ID, name, manufacturer, and the clock speed. This kind of monitoring is supported on most VM platforms.

```
public static char temperature_check()
{
    char result = 'n';
    ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("root\\WMI", "Select * From MSAcpi_ThermalZoneTemperature");
    try
    {
        int count = managementObjectSearcher.Get().Count;
    }
    catch (Exception ex)
    {
        bool flag = ex.Message == "Not supported ";
        if (flag)
        {
            result = 'a';
        }
    }
    return result;
}
```

Figure 22 - System Temperature check

Bypass:

When running in VM, the WMI request will return “Not Supported” error and the function will return ‘a’ to the caller. To bypass this, ‘result’ value should be patched to ‘n’.

Physical memory Anti-VM

Sandboxes, emulators and some VM’s are given limited RAM. Threat actors might add a check for not executing a malicious payload on a PC with limited RAM.

```
private static double get_physical_memory()
{
    ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("Select * From Win32_ComputerSystem");
    DateTime now = DateTime.Now;
    double num = 0.0;
    foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
    {
        ManagementObject managementObject = (ManagementObject)managementBaseObject;
        Program.ion00a53AkAbzOjJ7xwc68CICyByfJ4ZtZcxMTWBVyy5YWDHNhf2yFw1Tjr(now);
        num = Convert.ToDouble(managementObject["TotalPhysicalMemory"]);
    }
    return num / 1048576.0;
}
```

Figure 23 - Physical Memory Check

Bypass:

If your RAM memory is >4GB no bypass will be required, otherwise, 'num' value should be patched.

MAC Address Anti-VM

Vendors of different virtual environments hard code some values, MAC address is one of them. The malware might look for these values to detect a VM environment. Usually, it is enough to check the beginning of the MAC Address do detect a VM:

```
public static string get_mac_address()
{
    string text = "";
    foreach (NetworkInterface networkInterface in NetworkInterface.GetAllNetworkInterfaces())
    {
        bool flag = networkInterface.OperationalStatus == OperationalStatus.Up;
        if (flag)
        {
            text += networkInterface.GetPhysicalAddress().ToString();
            break;
        }
    }
    return text;
}
```

Figure 24 - MAC address check

Bypass:

To bypass this check, one can patch the 'text' value.

Packer

We used Goldfuscator v1.4 for file obfuscation in this stage.

Flag: {minerva_labs_ctf:bmlj-ZSBw-aWN0-dXJl-IGJybw}

Challenge 6 - Die-Hard

This is most likely the most confusing challenge of our CTF, and not because of the evasion techniques it uses, but because the of the way it is implemented. Every evasion technique that is not bypassed changes the Flag. Be careful! And patient!

Evasion Techniques

1. *QueryPerformanceCounter time Anti-Debugging*
2. *WMI PortConnector Anti-VM*
3. *HardDisk size Anti-VM*
4. *MAC Address Anti-VM*
5. *Stack Segment Register Anti-Debugging*
6. *GetLocalTime timing check*

QueryPerformanceCounter time Anti-debugging

As we mentioned before, when a process is traced in a debugger, there is a delay between instructions and execution. The delay between some parts of code can be measured and compared with the actual delay. One of the ways to check for time delay is by using QueryPerformanceCounter function, which according to Microsoft documentation “Retrieves the current value of the performance counter, which is a high resolution (<1us) time stamp that can be used for time-interval measurements.”

Figure 25 - QueryPerformanceCounter time Anti-Debugging

Bypass:

As in this check there is a comparison of a value in memory. We can change the value to be below or equal to the one it is compared to. Another way is to patch the value set in memory (from 1 to zero) or just change the returned value by this function to 0.

WMI PortConnector Anti-VM

Some research showed that querying **win32_portconnector** on VMs always returns null or empty. Malware might execute this query and check if the output is empty.

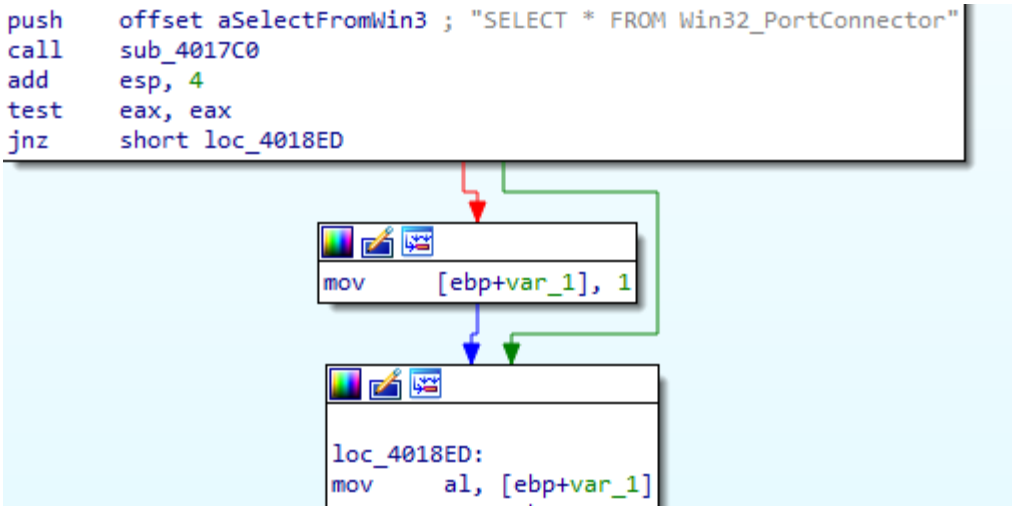

Figure 26 - WMI PortConnector Anti-VM

Bypass:

The best way to bypass this technique is to change the AL register at the end of the check function to 0.

HardDisk size Anti-VM

Some researchers prefer to run malicious files on “light weight” machines, which usually means a machine with limited hard disk space and limited memory. The Malware might check for hard disk storage size to detect anomalies that might point to running in VM:

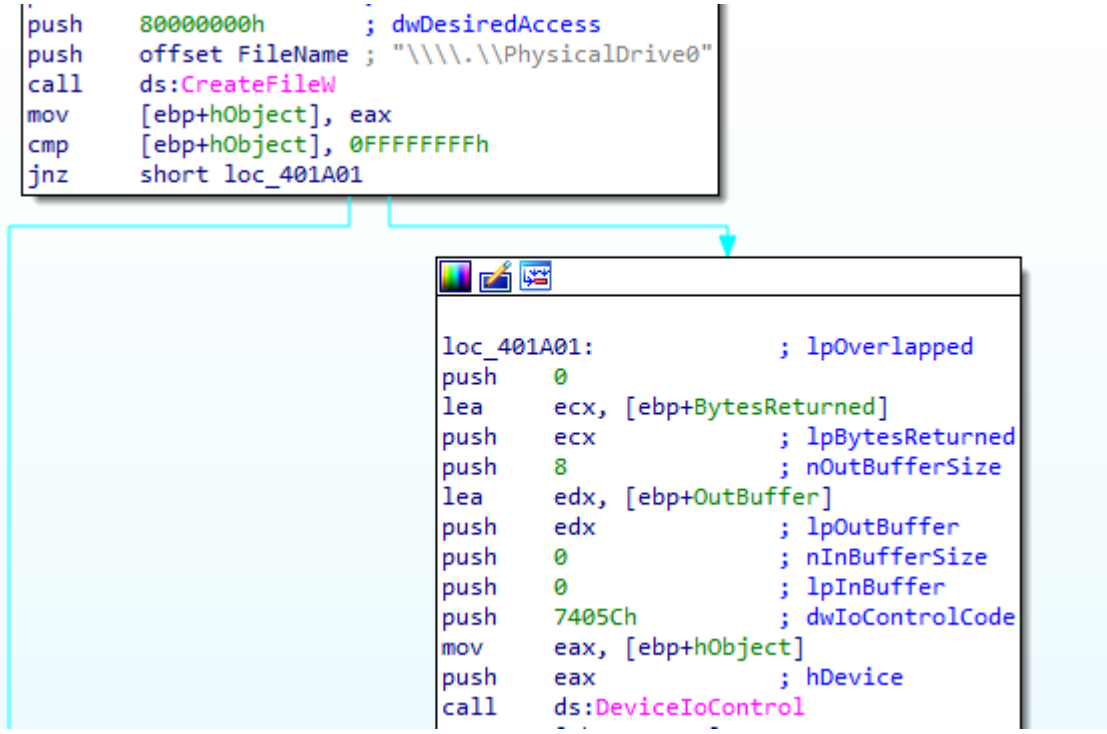

Figure 27 - part of Hard Disk size Anti-VM

Bypass:

If you didn’t build your machine with extremely low specs, you would not need to bypass this check, otherwise, the best way is to change the return value of this function to 0.

MAC Address Anti-VM

Vendors of different virtual environments hard code some values, MAC address is one of them. The malware might look for these values to detect a VM environment. The MAC address check might be implemented by using GetAdaptersInfo function which "retrieves adapter information for the local computer." Usually, it is enough to check the beginning of the MAC Address do detect a VM:

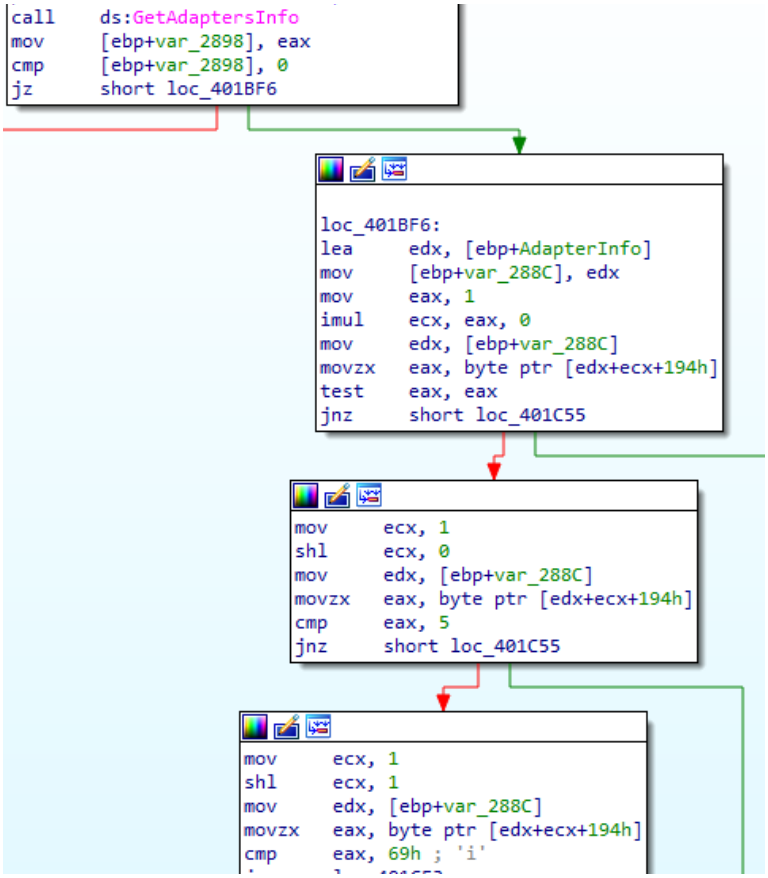

Figure 28 - part of MAC Address Anti-VM

Bypass:

There are two ways to bypass this check, the first is by changing MAC Address parts places in the EAX register so it will not match the known VM MAC addresses. The second is to change the function return value (EAX register) to 0.

Stack Segment Register Anti-Debugging

By tracing over the following sequence of assembly instructions the malware can detect if the program is being traced:

- push ss
- pop ss
- pushf

After single-stepping in a debugger through this assembly instructions, the Trap Flag will be set. Usually, a debugger clears the Trap Flag after each event, however, if we previously save EFLAGS to the stack (push ss), we'll be able to check whether the Trap Flag is set.

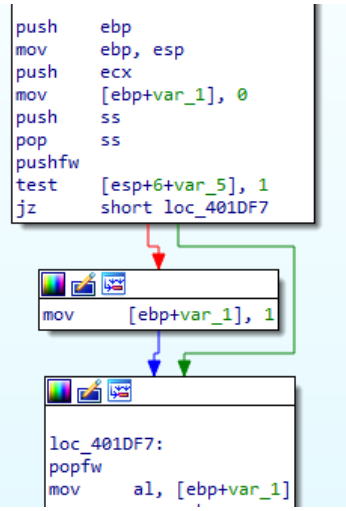

Figure 29 - Stack Segment Register Anti-Debugging

Bypass:

The Trap Flag is checked by the test instruction. We can bypass this technique by either patching the value in the stack or by changing a value of AL register at the end of the function.

GetLocalTime timing check

When a process is traced in a debugger, there is a delay between instructions and execution. The delay between some parts of code can be measured and compared with the actual delay. One of the ways to check for time delay is by using the GetLocalTime function which retrieves the current local date and time.

```
lea     eax, [ebp+SystemTime]
push    eax             ; lpSystemTime
call    ds:GetLocalTime
```

Figure 30 - part of GetLocalTime Anti-Debugging function

Bypass:

This check is implemented in a separate function which returns 1 if a debugger is detected, or 0 if not. The best way is to change the returned value by this function to 0.

Flag: {minerva_labs_ctf: R28g-SGFy-ZCBv-ciBH-byBI-b21lIQ}

Challenge 7- Help ME!

This challenge mostly covers the less known evasion techniques that are usually used by more sophisticated threat actors.

Evasion Techniques

1. *SetUnhandledException Anti-Debugging*
2. *PC Friendly Name Anti-VM*
3. *Poweshell WMI query Anti-VM*
4. *NtQuerySystemInformation Anti-Debugging*
5. *Int 2D Anti-Debugging*
6. *Heap Protection Anti-Debugging*

SetUnhandledException Anti-Debugging

By default, if an exception occurs and no exception handler is registered (or it is registered but doesn't handle such an exception), the UnhandledExceptionFilter function will be called. It is possible to register a custom unhandled exception filter using SetUnhandledExceptionFilter function. But, if the program is running under a debugger, the exception will be passed to the debugger and the custom filter won't be called. Therefore, if the unhandled exception filter is registered and the control is passed to it, then the process is not running with a debugger.

```
push    offset TopLevelExceptionFilter ; lpTopLevelExceptionFilter
call    ds:SetUnhandledExceptionFilter
```

Figure 31 - part of SetUnhandledExceptionFilter Anti-Debugging

Bypass:

There are two things that need the bypass here – the first is an INT3 instruction, which can be patched to NOP instruction(0x90) to skip by changing the EIP register to execute the next instruction. The second, is the return value from this function (which indicates if the file is executed in the debugger), this value should be changed to 0.

PC Friendly Name Anti-VM

Friendly names are assigned to complement unique identifiers that are comprised of numeric or alphanumeric code by default. The malware might retrieve a friendly name of all devices present on the current system by using SetupDiGetClassDevs, SetupDiEnumDeviceInfo and SetupDiGetDeviceRegistryProperty and check for the occurrence of the strings VMWARE, VBOX, VIRTUAL HD and QEMU within the name:

```
call    ds:SetupDiGetDeviceRegistryPropertyW
mov     ecx, [ebp+PropertyBuffer]
push    ecx             ; lpsz
call    ds:CharUpperW
push    offset SubStr   ; "VBOX"
mov     edx, [ebp+PropertyBuffer]
push    edx             ; Str
call    sub_401040
add     esp, 8
test    eax, eax
jz      short loc_40224A
```

```
loc_40224A:
push    offset aVmware  ; "VMWARE"
mov     eax, [ebp+PropertyBuffer]
push    eax             ; Str
```

Figure 32 - part of Friendly Name check function

Bypass:

There is a function that checks friendly name and returns 1 if the name contains the required string (such as "VMWARE") and 0 if the string was not found in friendly name. We might easily change the returned value from this function by changing the EAX register's value to 0.

Poweshell WMI query Anti-VM

Win32_ComputerSystem WMI class represents a computer system running Windows and contains a lot of basic system information. Several parameters returned from this query might indicate a VM. The malware can perform this check in many ways, we choose to execute a WMI query through PowerShell and save the output on disk. Later, the output will be checked for containing strings that might indicate a VM.

```
aPowershellExeW db 'powershell.exe -WindowStyle Hidden -command "gwmi -q ',27h,'selec'
db 't * from win32_computersystem',27h,' | Out-File -FilePath c:\temp'
db '\output.txt"',0
```

Figure 33 - Powershell Command Executed by the program

```
push    offset aCTempOutputTxt ; "C:\\temp\\output.txt"
call    pass_401540_0
lea     eax, [ebp+var_38]
push    eax
call    sub_402430
add     esp, 1Ch
mov     [ebp+var_8C], eax
;   try {
mov     [ebp+var_4], 0
mov     ecx, 1
imul    edx, ecx, 0
mov     [ebp+edx+Src], 56h ; 'V'
mov     eax, 1
shl     eax, 0
mov     [ebp+eax+Src], 4Dh ; 'M'
mov     ecx, 1
imul    edx, ecx, 3
mov     [ebp+edx+Src], 41h ; 'A'
mov     eax, 1
shl     eax, 1
mov     [ebp+eax+Src], 57h ; 'W'
mov     ecx, 1
shl     ecx, 0
mov     [ebp+ecx+var_18], 42h ; 'B'
mov     edx, 1
shl     edx, 1
mov     [ebp+edx+var_18], 4Fh ; 'O'
mov     eax, 1
shl     eax, 2
mov     [ebp+eax+Src], 52h ; 'R'
mov     ecx, 1
imul    edx, ecx, 5
mov     [ebp+edx+Src], 45h ; 'E'
mov     eax, 1
imul    ecx, eax, 6
```

Figure 34 - Output.txt check of containing VM strings

Bypass:

There are different ways to bypass this technique, one might change the output file content and remove all the strings indicating a VM, or there is an easier way by changing the return value from this function (EAX register) to 0.

NtQuerySystemInformation Anti-Debugging

By using NtQuerySystemInformation function it is possible to retrieve different types of information about the specified process. By querying a ProcessDebugPort(0x7) it is possible to retrieve the port number of the debugger. ProcessDebugPort class retrieves a DWORD value equal to 0xFFFFFFFF (decimal -1) if the process is being debugged.

Figure 35 - ProcessDebugPort retrieve

Bypass:

The best way to bypass this technique is to either patch the JMP (JNZ in our case) instruction or by changing the EIP.

Int 2D Anti-Debugging

When the Int 2D assembly instruction is executed, the exception EXCEPTION_BREAKPOINT is raised. But with INT2D, Windows uses the EIP register as an exception address and then increments the EIP register value.

This instruction can cause problems for some debuggers because after the EIP incrementation, the byte which follows the INT2D instruction will be skipped, and the execution might continue from the damaged instruction.

If the program is executed without a debugger, control will be passed to the exception handler.

```
int     2Dh             ; Windows NT - debugging services: eax = type
nop
mov     [ebp+var_19], 1
```

Bypass:

This check is implemented in a separate function which returns 1 if debugger is detected, or 0 if not. The best way is to change the returned value by this function to 0.

Heap Protection Anti-Debugging

If the HEAP_TAIL_CHECKING_ENABLED flag is set in NtGlobalFlag, the sequence 0xABABABAB will be appended (twice in 32-Bit and 4 times in 64-Bit Windows) at the end of the allocated heap block. The malware might look for these bytes to detect a debugger:

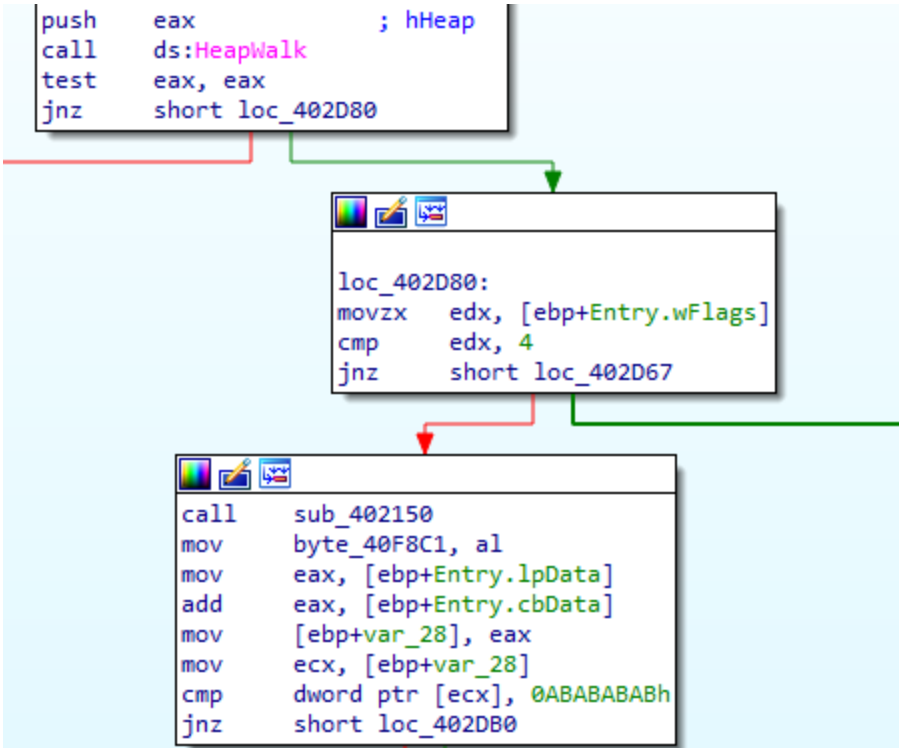

Figure 36 - Heap Protection Anti-Debugging

Bypass:

This check is implemented in a separate function which returns 1 if debugger is detected, or 0 if not. The best way is to change the returned value by this function to 0.

Flag: {minerva_labs_ctf:WW91-IGFy-ZSBh-bG1v-c3Qg-d2hlcmU}

Challenge 8- Shit! The Door

Introduction

The purpose of this challenge is to present a custom packer that might make a malicious executable undetectable by AV solutions, detect VMs and make a reverse engineer's life difficult.

Evasion Techniques

1. Custom Packer
2. IsDebuggerPresent – assembly implementation
3. Registry VM artifacts Anti-VM
4. CPUID Anti-VM
5. Parent process window name
6. Hardware Breakpoints Anti-Debugging
7. NtQuerySystemInformation Anti-Debugging

Custom Packer

The packer creates 4 new PE sections:

.tan – this section contains two RC4 keys - one XOR encoded and the other in plaintext. These will be used to decrypt the next payload.

.tnx – this section contains a shellcode that will execute before decrypting the payload. Usually it will perform some anti-debug\anti-vm checks and will decrypt an RC4 key from the ".tan" section if the execution environment is safe (no debugger or VM was found).

.mot – this is the encrypted main payload.

.pat – contains the packer stub, and new entry point.

There were three different shellcodes created that are stored in .tnx section. Shellcodes are added randomly every time the binary is packed. Shellcodes are responsible for the Anti-VM and Anti-Debugging checks.

IsDebuggerPresent – assembly implementation

An assembly implementation of IsDebuggerPresent API call.

```
mov     eax, large fs:30h
mov     [ebp+var_14], eax
sub     eax, edx
add     eax, 2
mov     eax, [ebp+var_14]
movsx   ecx, byte ptr [eax+2]
and     ecx, 1
jz      short loc_D5005F
```

Figure 37 - Debugger Flag value check

The ecx register holds a BeingDebbuged flag value.

Bypass:

The bypass is the same as of regular IsDebuggerPresent check, but in this case, we have to patch the ecx register.

Registry VM artifacts Anti-VM

The artifacts checked are:

- HKLM\HARDWARE\Description\System\BIOS\SystemProductName – the code queries the key value and checks if it contains 'VMWARE' or 'VBOX' strings.
- HKLM\HARDWARE\Description\System\BIOS\SystemBiosVersion – the code queries BIOS Version

CPUID Anti-VM

CPUID is an instruction-level detection method, which is mostly hard to detect. This instruction is executed with EAX=1 as input, the return value describes the processor's features. The 31st bit of ECX on a physical machine will be equal to 0. On a guest VM it will equal 1.

Bypass:

This check is performed in a separate function that returns 1 if running in VM and 0 if running on a physical machine. Patch the function return value in EAX register to bypass this technique.

Parent process window name

Some malwares might check their parent process name. Usually, user processes run under explorer.exe. So threat actors might check the parent process name and change the execution flow/exit if the parent process name is equal to one of the debugger names (IDA for example) or just different from what is expected:

Bypass:

When FindWindowA returns a non-zero value, you can patch it to 0, the search will run until all predefined debugger names are checked and if at the end of the list the process is not exited, the thread will finish its work and return to the main thread which will decrypt the flag.

Hardware Breakpoints Anti-Debugging - in different thread

Hardware breakpoints are a technology implemented by Intel in their processor architecture and are controlled using special registers known as Dr0-Dr7. Dr0 through Dr3 are 32-bit registers that hold the address of the breakpoint.

The main check based on GetThreadContext returns the value and it runs in a separate thread.

Bypass:

To bypass this check, the return value of GetThreadContext needs to be set to 0.

NtQuerySystemInformation Anti-Debugging

Threat actors might use NtQuerySystemInformation to try to retrieve a handle to the current process's debug object handle. If the function is successful, it will return true which means we're being debugged, or it will return false if it fails the process is not being debugged.

Bypass:

To bypass this Anti-Debugging technique, the return value from NtQuerySystemInformation needs to be patched.

Flag: {minerva_labs_ctf: U2Vl-IHlv-dSBu-ZXh0-IHll-YXI}