

Forum:[Hack Forums](#)

Thread: [understanding thread context in Runpe method](#)

invisible_hack2 - November 22, 2019, 1:43 pm

hi guys, i've some questions

why is ebx+8? it's eax register in context structure but why? is imagebase in eax register of context struct ? i don't get any documentation about it in internet

here is the code

```
Code ReadProcessMemory(Pi.hProcess, LPCVOID(CTX->Ebx + 8), LPVOID(&dwImageBase), 4, NULL);
```

[CONTEXT \(winnt.h\) - Win32 apps | Microsoft Docs](#)

full source

[PE-Crypter/runPE.h at master · jozemberi/PE-Crypter · GitHub](#)

gigajew - November 22, 2019, 2:25 pm

Don't focus on what's where in the stack, what you need to look at is how the windows loader works when a process is created to get an answer to your question

invisible_hack2 - November 22, 2019, 3:57 pm

Don't focus on what's where in the stack, what you need to look at is how the windows loader works when a process is created to get an answer to your question

where could i get that info?

Rossiya - November 23, 2019, 10:38 am

Process hollowing is nothing more than replacing another process's image with a different one - it's often used to bypass detection where applicable and mask rogue operations by having them be performed by what would appear to be a legitimate process that's actually been hijacked.

Simple & quick basics:

1. Create the process-to-be-hollowed in a suspended state.
2. Unmap the image of the process for the process-to-be-hollowed.
3. Map the new image to the process-to-be-hollowed.
4. Cleanup.
5. Resume execution for the main thread of the hollowed process.

Cleanup stage includes anything from address relocation to hijacking code execution to the new start address via registers with the thread context.

Furthermore, I'd recommend using system calls for `NtCreateUserProcess`, `NtGetContextThread`, `NtSetContextThread`, `NtUnmapViewOfSection`, `NtReadVirtualMemory`, `NtWriteVirtualMemory`, `NtResumeThread`, `NtClose`, etc.

If you decide to use `NtCreateUserProcess` then reverse-engineer `CreateProcessInternalW` to get a proper implementation working as `NtCreateUserProcess` alone is only for native PEs (Windows NT native to clarify).

I recommend you study Windows NT APIs (how they work and what their intended use is, as well as how to use them) as well as the Windows PE File Format before you continue.

LynxAxe - November 23, 2019, 5:51 pm

(Nov 23, 2019 11:38 AM)Rossiya Wrote: [\[CITE\]](#) 2. Unmap the image of the process for the process-to-be-hollowed.

It's better for you to use `NtUnmapViewOfSectionEx` instead of `NtUnmapViewOfSection` because these NTDLL `Ex*` APIs keep getting ignored or forgotten about by Anti-Virus products that are using user-mode hooking.

I've been told that `NtUnmapViewOfSection/Ex` is destined for doom on some Windows PCs so I would suggest you rock your own emulation of it!

HFDone - November 24, 2019, 2:43 am

at the end the compiler translate your code to assembly

to understand it you should look into asm runpe

the important part is the end when you can see why you need the context struct and how its done via thread

```
...
invoke GetThreadContext,[_PI.hProcess],CTX
invoke WriteProcessMemory,[_PI.hProcess],dword[CTX.Ebx+8],imagebase,0x4,0
pop eax
add eax,dword[eax+0x3C]
mov eax,dword[eax+0x28]
mov [CTX.Eax],eax ; EntryPoint
invoke SetThreadContext,[_PI.hProcess],CTX
invoke ResumeThread,[_PI.hThread]
```

invisible_hack2 - November 24, 2019, 3:03 am

(Nov 23, 2019 11:38 AM)Rossiya Wrote: [\[CITE\]](#) Process hollowing is nothing more than replacing another process's image with a different one - it's often used to bypass detection where applicable and mask rogue operations by having them be performed by what would appear to be a legitimate process that's actually been hijacked.

Simple & quick basics:

1. Create the process-to-be-hollowed in a suspended state.
2. Unmap the image of the process for the process-to-be-hollowed.
3. Map the new image to the process-to-be-hollowed.
4. Cleanup.
5. Resume execution for the main thread of the hollowed process.

Cleanup stage includes anything from address relocation to hijacking code execution to the new start address via registers with the thread context.

Furthermore, I'd recommend using system calls for `NtCreateUserProcess`, `NtGetContextThread`, `NtSetContextThread`, `NtUnmapViewOfSection`, `NtReadVirtualMemory`, `NtWriteVirtualMemory`, `NtResumeThread`, `NtClose`, etc.

If you decide to use `NtCreateUserProcess` then reverse-engineer `CreateProcessInternalW` to get a proper implementation working as `NtCreateUserProcess` alone is only for native PEs (Windows NT native to clarify).

I recommend you study Windows NT APIs (how they work and what their intended use is, as well as how to use them) as well as the Windows PE File Format before you continue. Thank you, it's exactly the kind of answer I was looking for

(Nov 24, 2019 03:43 AM)HFDone Wrote: [\[CITE\]](#) at the end the compiler translate your code to assembly
to understand it you should look into asm runpe
the important part is the end when you can see why you need the context struct and how its done via thread

```
...  
invoke GetThreadContext,[_PI.hProcess],CTX  
invoke WriteProcessMemory,[_PI.hProcess],dword[CTX.Ebx+8],imagebase,0x4,0  
pop eax  
add eax,dword[eax+0x3C]  
mov eax,dword[eax+0x28]  
mov [CTX.Eax],eax ; EntryPoint  
invoke SetThreadContext,[_PI.hProcess],CTX  
invoke ResumeThread,[_PI.hThread]  
so basically eax register is where entry point start ?
```

HFDone - November 24, 2019, 7:35 am

[WriteProcessMemory function \(memoryapi.h\) - Win32 apps | Microsoft Docs](#)

`lpBaseAddress`

A pointer to the base address in the specified process to which data is written. Before data transfer occurs, the system verifies that all data in the base address and memory of the specified size is accessible for write access, and if it is not accessible, the function fails.

`lpBuffer`

A pointer to the buffer that contains data to be written in the address space of the specified process.

`Ebx+8 ; ImageBase`

`eax ; EntryPoint`

LynxAxe - November 24, 2019, 8:06 am

(Nov 23, 2019 06:51 PM)LynxAxe Wrote: [\[CITE\]](#) It's better for you to use `NtUnmapViewOfSectionEx` instead of `NtUnmapViewOfSection` because these NTDLL `Ex*` APIs keep getting ignored or forgotten about by Anti-Virus products that are using user-mode hooking.

I've been told that `NtUnmapViewOfSection/Ex` is destined for doom on some Windows PCs so I would suggest you rock your own emulation of it!

Bear in mind, `NtUnmapViewOfSectionEx` was only introduced for Windows 8, so make sure you check the version of Windows that is installed on the PC before trying to use it.

```
Code NTSYSAPI NTSTATUS NtUnmapViewOfSection(  
    HANDLE ProcessHandle,  
    PVOID BaseAddress  
);
```

To ensure compatibility, you should use dynamic address resolving (roll your own better emulation of GetProcAddress with EAT scanning) so you can target one or the other depending on whether NtUnmapViewOfSectionEx is present on the version of Windows that's running.

Another member also suggested using system calls manually and I vouch for this suggestion and would recommend it instead of finding the address of those APIs from NTDLL and using them as-is, I cannot express the importance of this !

Zephyros - November 24, 2019, 1:15 pm

At the process entry, before code is executed, the ebx register contains the PEB (process environment block)

It contains various important info, but in this case it is used to read and later write the processes image base. (8 is the offset from the start of PEB)

Updating it to the correct value is important if the injected Exe contains resources, else the Windows APIs can't find them.

For more info: [x86 - Windows initial execution context - Stack Overflow](#)

invisible_hack2 - November 25, 2019, 12:05 pm

(Nov 24, 2019 08:35 AM)HFDone Wrote: [\[CITE\] WriteProcessMemory function \(memoryapi.h\) - Win32 apps | Microsoft Docs](#)

lpBaseAddress

A pointer to the base address in the specified process to which data is written. Before data transfer occurs, the system verifies that all data in the base address and memory of the specified size is accessible for write access, and if it is not accessible, the function fails.

lpBuffer

A pointer to the buffer that contains data to be written in the address space of the specified process.

Ebx+8 ; ImageBase

eax ; EntryPoint

Good explanation ,now everything is much clearer

(Nov 24, 2019 02:15 PM)Zephyros Wrote: [\[CITE\]](#) At the process entry, before code is executed, the ebx register contains the PEB (process environment block)

It contains various important info, but in this case it is used to read and later write the processes image base. (8 is the offset from the start of PEB)

Updating it to the correct value is important if the injected Exe contains resources, else the Windows APIs can't find them.

For more info: [x86 - Windows initial execution context - Stack Overflow](#)

Thank you too ? ?