

Zloader Strikes Back

Published: 2024-02-14 · Archived: 2026-04-05 13:01:37 UTC

Recently, we came across an update from [PolySwarm](#) regarding a new Variant of Zloader. Zloader is a malware based on Zeus, which has been targeting financial institutions and its customers. This blog gets into the nuances of the new techniques used by Zloader.

Technical Analysis

It was observed that Zloader had very few Import functions and it was obfuscated and threat actors were making sure that Zloader only runs with the filename “IonPulse.exe”.

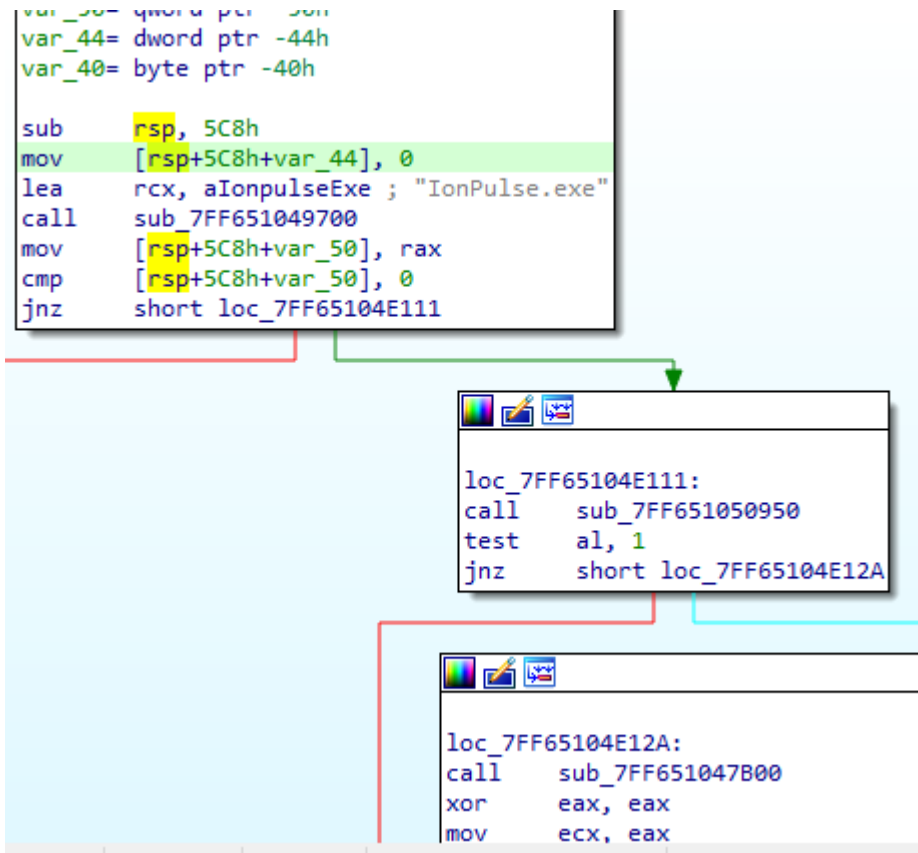


Figure 1: Precheck before running

Once it checks that the name is IonPulse.exe, it gets the handle of Ntdll.dll using CreateFileA.

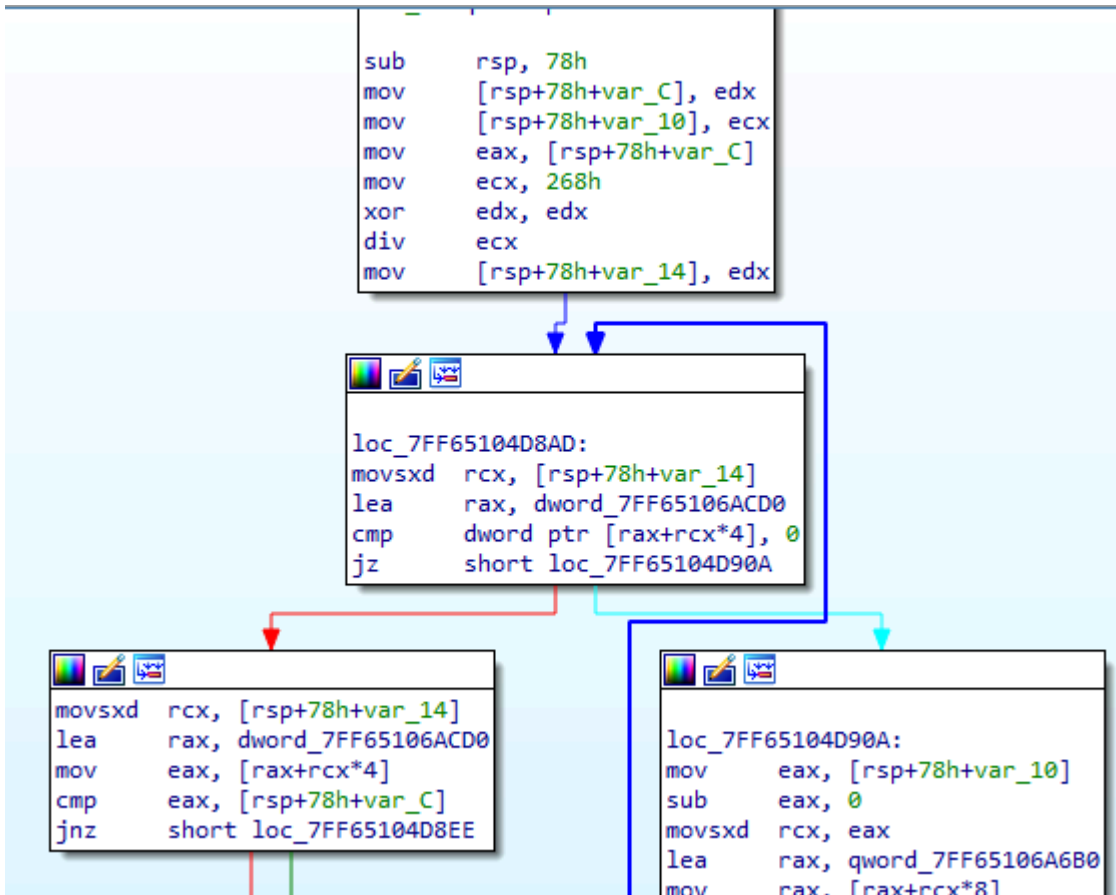


Figure 2: Mapping API with hashes

It is making use of the above mentioned Function in Figure 2 to resolve the API.

```

var_18= qword ptr -18h
var_10= qword ptr -10h
var_1= byte ptr -1

sub    rsp, 98h
mov    [rsp+98h+var_10], rdx
mov    [rsp+98h+var_18], rcx
mov    rcx, [rsp+98h+var_18]
call   sub_7FF651064450
mov    [rsp+98h+var_38], rax
xor    ecx, ecx
mov    edx, 0B3E383DFh
call   sub_7FF651061110
mov    rcx, [rsp+98h+var_38]
mov    edx, 80000000h
mov    r8d, 1
xor    r9d, r9d
xor    r10d, r10d
mov    dword ptr [rsp+98h+var_78], 3
mov    [rsp+98h+var_70], 0
mov    [rsp+98h+var_68], 0
call   rax
mov    [rsp+98h+var_20], rax
mov    rax, 0FFFFFFFFFFFFFFFh
cmp    [rsp+98h+var_20], rax
int3  short loc_7FF65104D776
    
```

Figure 3: CreateFileA

It gets the handle of Ntdll.dll using CreateFileA.

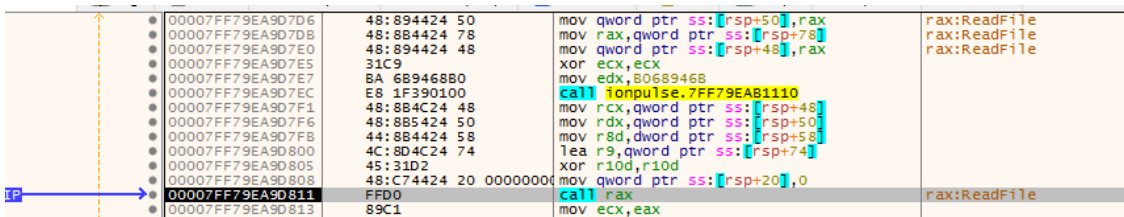


Figure 4: Reading ntdll

Then uses ReadFile to copy the contents of Ntdll.dll. Before doing that it allocates memory using VirtualAlloc.

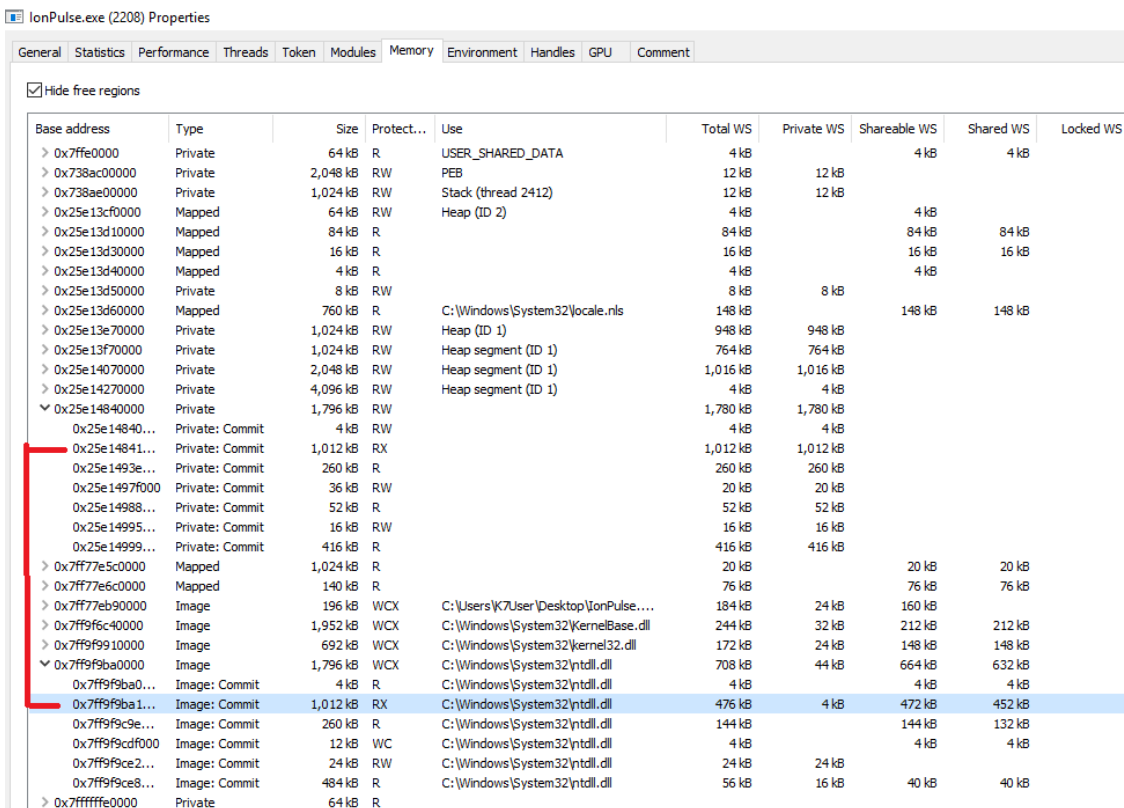


Figure 5: Ntdll.dll copied

Above figure shows the copied content of Ntdll.dll.

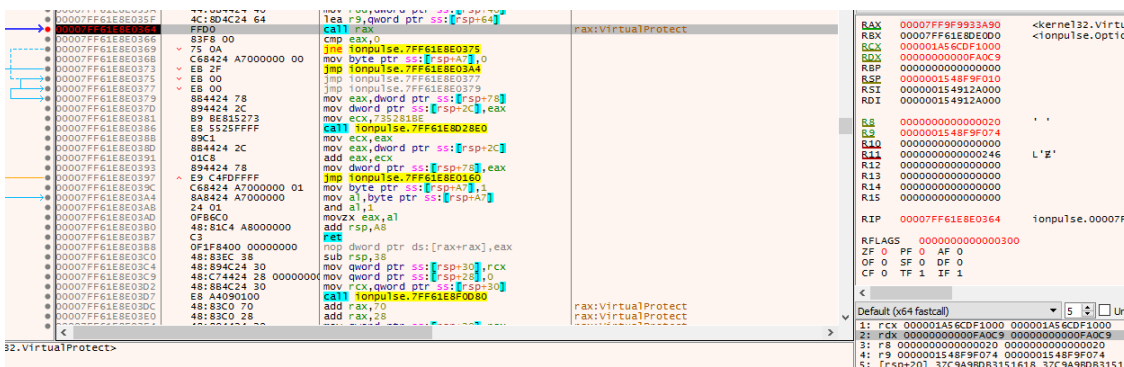


Figure 6: VirtualProtect

After copying Ntdll.dll it is using VirtualProtect to change the memory protection accordingly.

```

00007FF721D8BD58 BA A4F36B7 mov edx,B736AFA4
00007FF721D8BD5D E8 AE530100 call ionpulse.7FF721DA1110
00007FF721D8BD62 4C:888C24 90000000 mov r9,qword ptr ss:[rsp+90]
00007FF721D8BD6A 4C:889C24 A0000000 mov r11,qword ptr ss:[rsp+A0]
00007FF721D8BD72 4C:889424 B0000000 mov r10,qword ptr ss:[rsp+B0]
00007FF721D8BD7A 48:8D8C24 88010000 lea rcx,qword ptr ss:[rsp+188]
00007FF721D8BD82 48:8D9424 C8010000 lea rdx,qword ptr ss:[rsp+1C8]
00007FF721D8BD8A 45:31C0 xor r8d,r8d
00007FF721D8BD8D 4C:895C24 20 mov qword ptr ss:[rsp+20],r11
00007FF721D8BD92 48:C74424 28 00000000 mov qword ptr ss:[rsp+28],0
00007FF721D8BD98 48:C74424 30 00000000 mov qword ptr ss:[rsp+30],0
00007FF721D8BDA4 4C:895424 38 mov qword ptr ss:[rsp+38],r10
00007FF721D8BDA9 48:C74424 40 00000000 mov qword ptr ss:[rsp+40],0
00007FF721D8BDB2 48:C74424 48 00000000 mov qword ptr ss:[rsp+48],0
00007FF721D8BDB8 C74424 50 01000000 mov dword ptr ss:[rsp+50],1
00007FF721D8BDC3 FFD0 call rax
00007FF721D8BDC5 89C1 mov ecx,eax
00007FF721D8BDC7 31D2 xor edx,edx
00007FF721D8BDC9 E8 F2910000 call ionpulse.7FF721D94FC0
00007FF721D8BDCD A9 01000000 test eax,1
00007FF721D8BDD3 75 18 jne ionpulse.7FF721D8BDED
    
```

Figure 7: Creating msieexec.exe

It is making use of RtlInitUnicodeString, RtlCreateProcessParametersEx to create a structure which can be used by NtCreateUserProcess later. Then it make use of Associated syscall to NtCreateUserProcess to run msieexec.exe.

```

e:\exe - Thread: Main Thread 464 - v64dbg
000002809ED55475 CD 2E ret
000002809ED55477 C3 ret
000002809ED55478 0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
000002809ED55480 4C:8801 mov r10,rcx
000002809ED55483 B8 3A000000 mov eax,3A
000002809ED55488 F60425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
000002809ED55490 75 03 jne 2809ED55495
000002809ED55492 0F05 syscall
000002809ED55494 C3 ret
000002809ED55495 CD 2E ret
000002809ED55497 C3 ret
    
```

Figure 8: Syscall

It was making use of Syscall to Write into msieexec.exe and had allocated memory before doing that. This syscall is related to NtWriteVirtualMemory which is Similar to WriteProcessMemory in WinAPI.

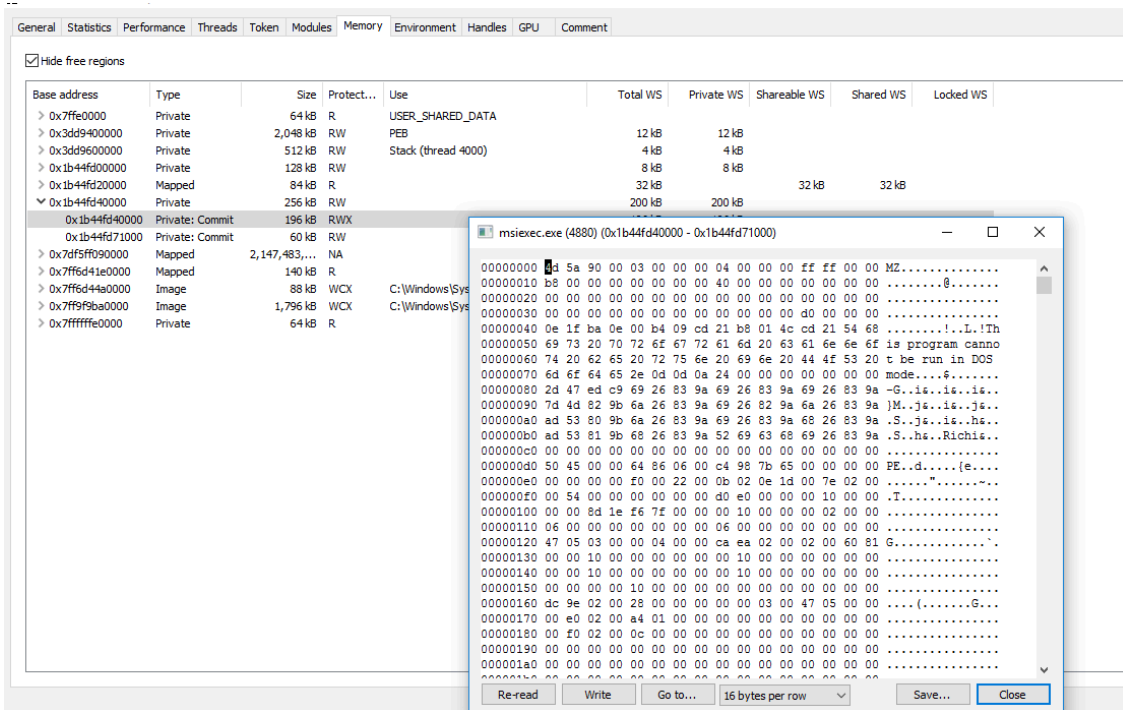


Figure 9: Zloader injected in msieexec.exe

Then makes use of another syscall to the adjacent function of NtProtectVirtualMemory, to change its memory protection to 'Execute'. Along with that it will use Syscall associated with NtGetContextThread, NtSetContextThread and NtResumeThread. Doing this it is hijacking the Thread.

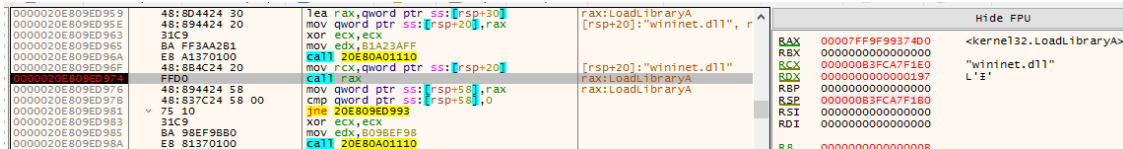


Figure 10: Loading wininet.dll

It will then load wininet.dll and ws2_32.dll using LoadLibraryA to connect to C2.

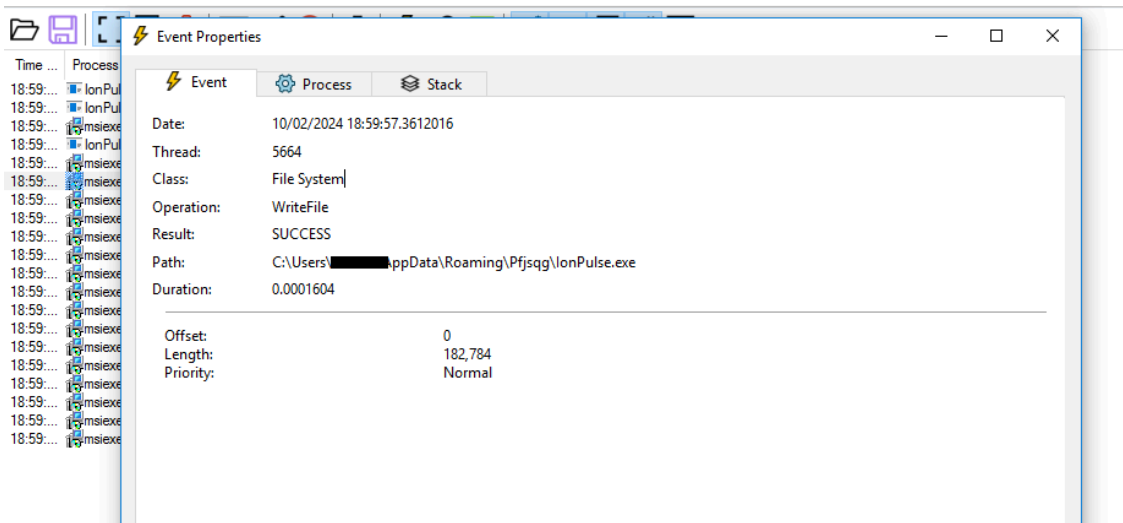


Figure 11: Self Copy

It will then make a self Copy in AppData\Roaming

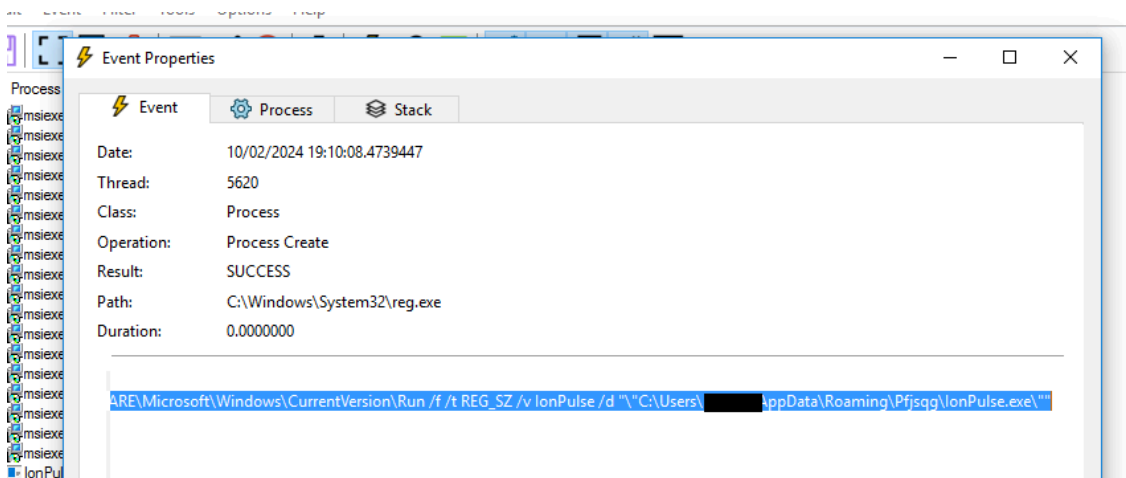


Figure 12: Run Entry

Persistence is ensured through the Run registry and msisec.exe starts connecting to C2 and then IonPulse.exe exits.

By this we can see that Zloader has started using Syscall for evasion, along with loading new Ntdll.dll.

We at K7 Labs provide detection for Zloader and all the latest threats. Users are advised to use a reliable security product such as “K7 Total Security” and keep it up-to-date to safeguard their devices.

Indicators of Compromise (IOCs)

FileName	Hash	Detection Name
IonPulse.exe	71C72AD0DA3AF2FCA53A729EF977F344	Trojan (005afb2c1)

References

<https://www.zscaler.com/blogs/security-research/zloader-no-longer-silent-night>

<https://captmeelo.com/redteam/maldev/2022/05/10/ntcreateuserprocess.html>

Source: <https://labs.k7computing.com/index.php/zloader-strikes-back/>