# Approaches to Reverse Engineering Dotnet Applications

A survey of tools, techniques, and countermeasures

# Agenda

- Goals of Server-side application reverse engineering
- Server-side Dotnet
- Client-side Dotnet
- Modifying compile CLR bytecode without source code
- Anti-reverse engineering techniques

# New presentation, who dis?

Nick Starke

Threat Researcher at Aruba Threat Labs within the Office of the CTO at Aruba Networks / Hewlett Packard Enterprise

- Focused on firmware security, especially in networking appliances
- Board member of SecDSM (https://secdsm.org)
- Lives in Bondurant!
- Moved into Security from Web Development
- Blog: https://nstarke.github.com
- Bandcamp: https://nstarke.bandcamp.com

# TL;DR

- For applications that compile down to byte code (JVM / CLR, primarily) there are tools that can take a compiled dll, jar, war, exe and create a near-source code quality representation of the code.
- There are ways to modify a compiled application without source code.
  - Code signing helps mitigate the risk of this type of attack
- Obfuscation is usually enough of an impediment for Reverse Engineers

# Why reverse engineer server-side applications? - Security

- As an attacker, often compiled applications contain secrets like keys and passwords
- As an attacker, you might want to modify an application without the source code
  - This is possible using tools like ILASM.exe/ILDASM.exe for dotnet CLR

# Why reverse engineer server-side applications? - Dev

- Have you lost the source code? Data loss does happen :-(
- As a developer, you may need to integrate with a product that has no documentation (legacy code, anyone?)
- As a developer, you may want to analyze proprietary code to understand how it works
- As a developer, it is important to understand what an attacker can do with your production binaries from a security perspective

# Dotnet

- .cs files compile down to .dll or .exe files
- Based on MSIL bytecode for server side apps
  - The equivalent of Java’s JVM Bytecode / Smali Bytecode

# Dotnet

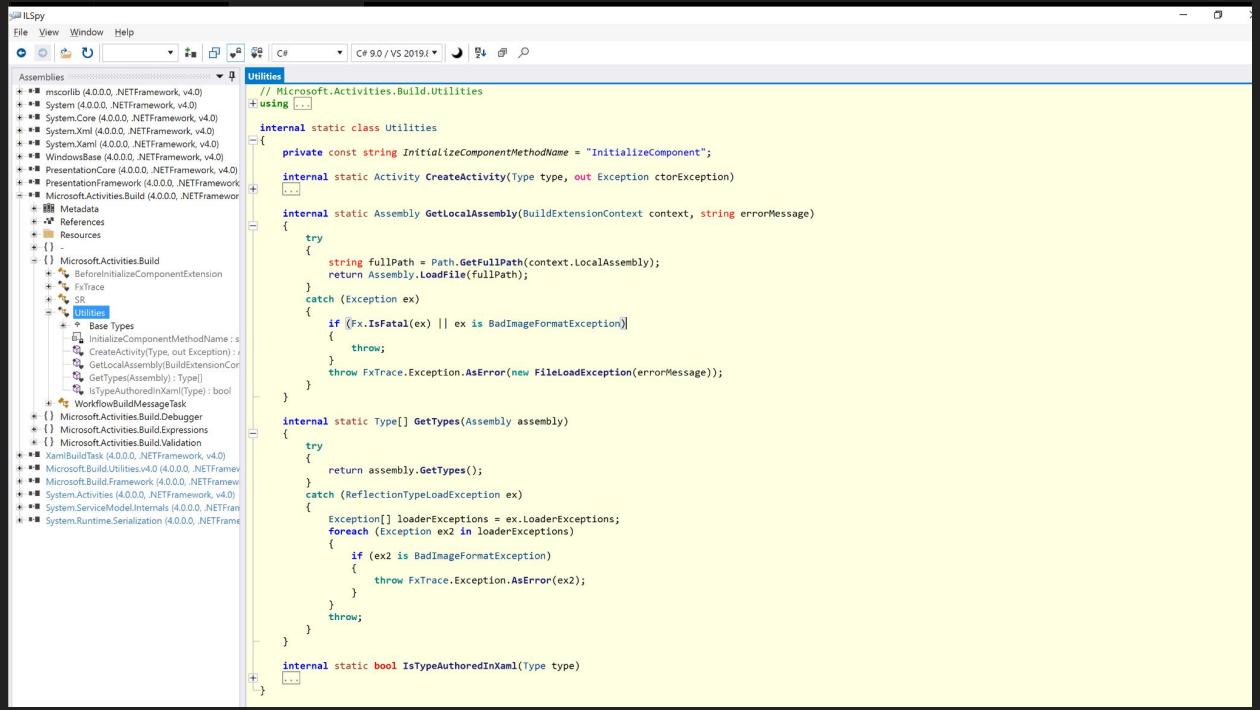
- Compiles down to MSIL (Microsoft Intermediate Language)
  - The .NET equivalent of JVM Bytecode
- This runs on the .NET CLR (Common language runtime)
- Source files are .cs files which compile to exe or dll
  - DLL’s more common for web apps

# Dotnet - ILSpy

ILSpy - https://github.com/icsharpcode/ILSpy

- Open source
- Can run on Linux/MacOS/Windows
- Sufficient Output

# ILSpy Screenshot



# Dotnet  - dotPeek

dotPeek - https://www.jetbrains.com/decompiler/

- JetBrains dotnet Decompiler
- Closed Source
- Free to use
- Can attempt to export DLL / EXE files as visual studio projects for recompilation

# Dotpeek Screenshot

JetBrains dotPeek

File View Navigate Inspect Tools Windows Help

Assembly Explorer

Type to search

- BIOSiQUtils (5.4.1.14954, msil, .Net Framework v4.5)
  - Metadata
  - References
    - log (5.4.1.14954)
    - ManagedWMI (5.4.1.14954)
    - mscorlib (4.0.0.0)
    - Newtonsoft.Json (12.0.0.0)
    - System (4.0.0.0)
  - BIOSiQ
    - CBIOSiQTelemetryEventPosting
    - CBIOSIQUtils
    - CBIQJSONParser
    - IBIOSiQTelemetryEventPosting
  - TimerWakesUp
- ManagedWMI (5.4.1.14954, msil, .Net Framework v4.5)
- mscorlib (4.0.0.0, x64)
- MSI Reminder (1.10.1707.1901, msil, .Net Framework v4.0)
- MSIOnlineRegister (1.10.1707.1901, msil, .Net Framework v4.0
- scmfb (1.10.1707.1901, msil, .Net Framework v4.0)
- scmreg (1.0.1512.401, msil, .Net Framework v4.0)
- System.Core (4.0.0.0, msil, .Net Framework v4.0)
- System.Data (4.0.0.0, x64, .Net Framework v4.0)
- System (4.0.0.0, msil, .Net Framework v4.0)
- System.Web (4.0.0.0, x64, .Net Framework v4.0)
- System.Xml (4.0.0.0, msil, .Net Framework v4.0)

CBIOSiQTelemetryEventPosting.cs

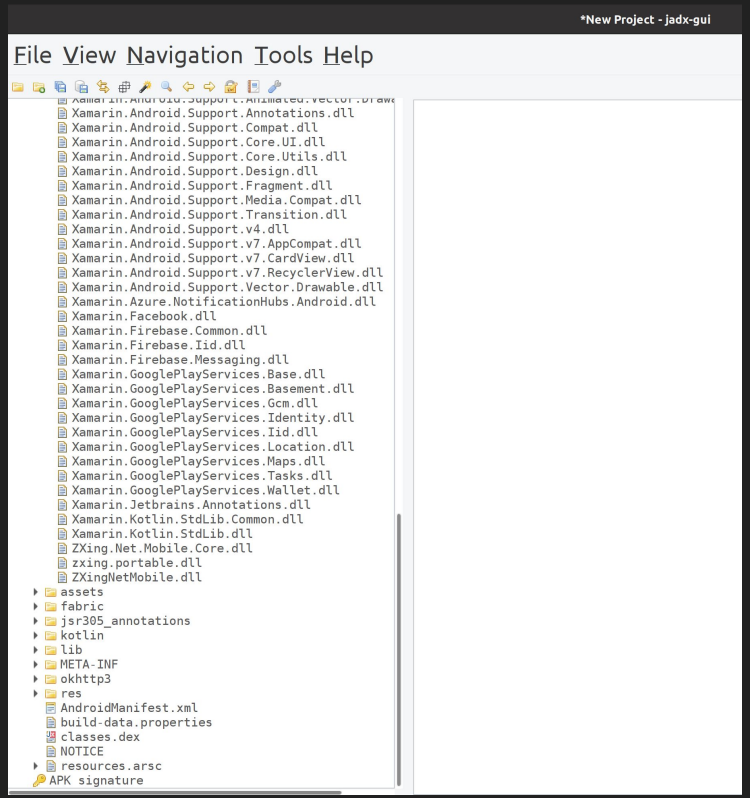
```
// Decompiled with JetBrains decompiler
// Type: BIOSiQ.CBIOSiQTelemetryEventPosting
// Assembly: BIOSiQUtils, Version=5.4.1.14954, Culture=neutral, PublicKeyToken=null
// MVID: 04D17FCD-D4C5-475E-9374-10859E23AE68
// Assembly location: C:\Program Files\Dell\SARemediation\agent\BIOSiQUtils.dll

using LogWrapper;
using System.Collections.Generic;
using TimerWakesUp;

namespace BIOSiQ
{
  public class CBIOSiQTelemetryEventPosting : ITimerWakesUp
  {
    private const int DefTimeroutInMs = 900000;
    private bool _shouldSendEvents;

    public CBIOSiQTelemetryEventPosting(
      IBIOSiQTelemetryEventPosting observer,
      bool shouldSendEvents = true)
    {
      Logger.Debug(string.Format("BIOSiQEvent CBIOSiQTelemetryEventPosting() IN"));
      this.m_observer = observer;
      this._shouldSendEvents = shouldSendEvents;
      Logger.Info("BIOSiQEvent, if we will send BIOSiQ event: " + this._shouldSendEvents.ToString());
      this.m_timer = new CTimerWakesUp((ITimerWakesUp) this);
      if (this.m_timer != null)
      {
        Logger.Debug(string.Format("BIOSiQEvent CBIOSiQTelemetryEventPosting() IN1"));
        this.m_timer.Restartup(900000);
      }
      Logger.Debug(string.Format("BIOSiQEvent CBIOSiQTelemetryEventPosting() OUT"));
    }

    public bool handleTimerWakesUp(object sender, object userToken)
    {
      Logger.Debug(string.Format("BIOSiQEvent handleTimerWakesUp() IN"));
      this.m_timer = (CTimerWakesUp) null;
      List<List<KeyValuePair<string, string>>> info = CBIOSIQUtils.GetInfo();
      List<List<KeyValuePair<string, string>>> alert = CBIOSIQUtils.GetAlert();
      Logger.Debug(string.Format("BIOSiQEvent handleTimerWakesUp() IN1"));
      if (this.m_observer != null)
      {
        if (info != null && info.Count > 0)
        {
          if (this._shouldSendEvents)
          {
            foreach (List<KeyValuePair<string, string>> keyValuePairs in info)
```

# Client-side Dotnet

Jadx-gui - https://github.com/skylot/jadx

- Useful for extracting Xamarin Assemblies
- Extracts the static content (res/) from the APK and presents it in a tree view

# JADX-GUI - Xamarin Disassemblies



# Modifying MSIL Bytecode without Source Code

The next few slides will focus on techniques for modifying MSIL Bytecode without access to the original source code.

We'll discuss:

- Why would anyone want to do this?
- Examples
- Process
- Tooling

# Why would anyone want to do this?

Development:

- Modify an application when source code is lost

Security:

- Patch an application to log out sensitive information

# Examples of Patching MSIL Bytecode - Security

Server-side

- Server side: when a login request is received, log out the username and password to a file on the filesystem.

Client-side

- Client side: make an HTTP request to an unauthorized remote server with authentication tokens received from a legitimate authentication request

# Process

1) Write out Dotnet code you wish to inject in a console application.  Create a function that accepts the data you wish to operate on.
2) Disassemble this console application
3) Disassemble the source code you wish to inject code into
4) Modify the source code disassembly to include the console application disassembly and write integration disassembly to call the function you wrote in 1)
5) Reassemble source .cs file
6) Reassemble DLL / Drop on file system cache.

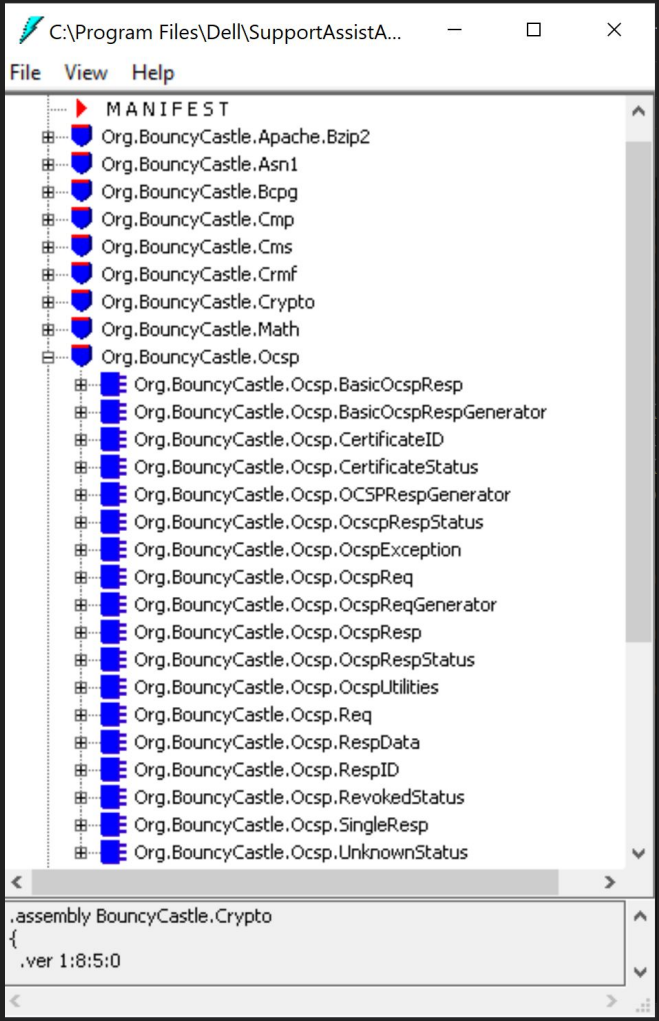
# Dotnet Disassembler / Assembler Duo

Dotnet MSIL Assembler: ILASM.exe - https://docs.microsoft.com/en-us/dotnet/framework/tools/ilasm-exe-il-assembler

Dotnet MSIL Disassembler: ILDASM.exe - https://docs.microsoft.com/en-us/dotnet/framework/tools/ildasm-exe-il-disassembler

- These two tools are built to work with each other.
- Available through Visual Studio Developer Shell

# ILDASM


C:\Program Files\Dell\SupportAssistA...
File
View
Help
MANIFEST
Org.BouncyCastle.Apache.Bzip2
Org.BouncyCastle.Asn1
Org.BouncyCastle.Bcpg
Org.BouncyCastle.Cmp
Org.BouncyCastle.Cms
Org.BouncyCastle.Crmf
Org.BouncyCastle.Crypto
Org.BouncyCastle.Math
Org.BouncyCastle.Ocsp
Org.BouncyCastle.Ocsp.BasicOcspResp
Org.BouncyCastle.Ocsp.BasicOcspRespGenerator
Org.BouncyCastle.Ocsp.CertificateID
Org.BouncyCastle.Ocsp.CertificateStatus
Org.BouncyCastle.Ocsp.OCSPRespGenerator
Org.BouncyCastle.Ocsp.OcscpRespStatus
Org.BouncyCastle.Ocsp.OcspException
Org.BouncyCastle.Ocsp.OcspReq
Org.BouncyCastle.Ocsp.OcspReqGenerator
Org.BouncyCastle.Ocsp.OcspResp
Org.BouncyCastle.Ocsp.OcspRespStatus
Org.BouncyCastle.Ocsp.OcspUtilities
Org.BouncyCastle.Ocsp.Req
Org.BouncyCastle.Ocsp.RespData
Org.BouncyCastle.Ocsp.RespID
Org.BouncyCastle.Ocsp.RevokedStatus
Org.BouncyCastle.Ocsp.SingleResp
Org.BouncyCastle.Ocsp.UnknownStatus
.assembly BouncyCastle.Crypto
{
.ver 1:8:5:0

# ILDASM - Disassembly

Org.BouncyCastle.Apache.Bzip2.CRC::UpdateCRC : void(int32)

Find Find Next

```
.method assembly hidebysig instance void
        UpdateCRC(int32 inCh) cil managed
{
  // Code size       47 (0x2f)
  .maxstack  4
  .locals init (int32 V_0)
  IL_0000:  ldarg.0
  IL_0001:  ldfld      int32 Org.BouncyCastle.Apache.Bzip2.CRC::globalCrc
  IL_0006:  ldc.i4.s   24
  IL_0008:  shr
  IL_0009:  ldarg.1
  IL_000a:  xor
  IL_000b:  stloc.0
  IL_000c:  ldloc.0
  IL_000d:  ldc.i4.0
  IL_000e:  bge.s      IL_0018
  IL_0010:  ldc.i4     0x100
  IL_0015:  ldloc.0
  IL_0016:  add
  IL_0017:  stloc.0
  IL_0018:  ldarg.0
  IL_0019:  ldarg.0
  IL_001a:  ldfld      int32 Org.BouncyCastle.Apache.Bzip2.CRC::globalCrc
  IL_001f:  ldc.i4.8
  IL_0020:  shl
  IL_0021:  ldsfld     int32[] Org.BouncyCastle.Apache.Bzip2.CRC::crc32Table
  IL_0026:  ldloc.0
  IL_0027:  ldelem.i4
  IL_0028:  xor
  IL_0029:  stfld      int32 Org.BouncyCastle.Apache.Bzip2.CRC::globalCrc
  IL_002e:  ret
} // end of method CRC::UpdateCRC
```

# How to mitigate this threat

- Strong name signing - https://docs.microsoft.com/en-us/dotnet/standard/assembly/sign-strong-name
- Read only file system for executable code

Android implements code signing by default - consider this for your production applications even when they are server-side.

# Anti-reverse engineering techniques

Obfuscation!

- Dotfuscator - https://www.preemptive.com/products/dotfuscator

Benefits:

- Makes code extremely difficult to reverse
- Makes code extremely difficult to modify

Cons:

- Server-side: usually expensive in terms of $ cost

# Goals of Obfuscation

Obfuscation can be used to deter attackers

Usually all you need to do is put up enough of a barrier to entry that it makes a potential attacker move on to the next target

Obfuscation alone is not sufficient to secure an application!

- Secrets should not be stored in source code
- Secrets should not be stored in source code
- SECRETS SHOULD NOT BE STORED IN SOURCE CODE

# Going Further

- Managed Code Rootkits (Book): https://www.amazon.com/Managed-Code-Rootkits-Hooking-Environments/dp/1597495743

# Thank you!

Questions?

Contact:

https://twitter.com/nstarke

- Blog: https://nstarke.github.com
- Bandcamp: https://nstarke.bandcamp.com