Introduction to Reverse Engineering Server-Side Applications for Web Developers

A brief survey of tools and techniques

Agenda

- Goals of Server-side application reverse engineering
- Java
- Dotnet
- JavaScript (Briefly)
- 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.
- Except for one tool, this code cannot be recompiled from the tool output.
- There are ways to modify a compiled application without source code.
- For applications written in interpreted languages (python, ruby, javascript) there is no compiled code (usually) so Reverse engineering becomes a code-review exercise
- 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 (watttttttt)
  - This is possible using tools like ILDASM/ILASM for .NET and Jasper/Jasmin for Java
  - However, it is not possible for the most part with the tools presented today
  - We won’t cover this in any detail in this presentation :-)

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

Java

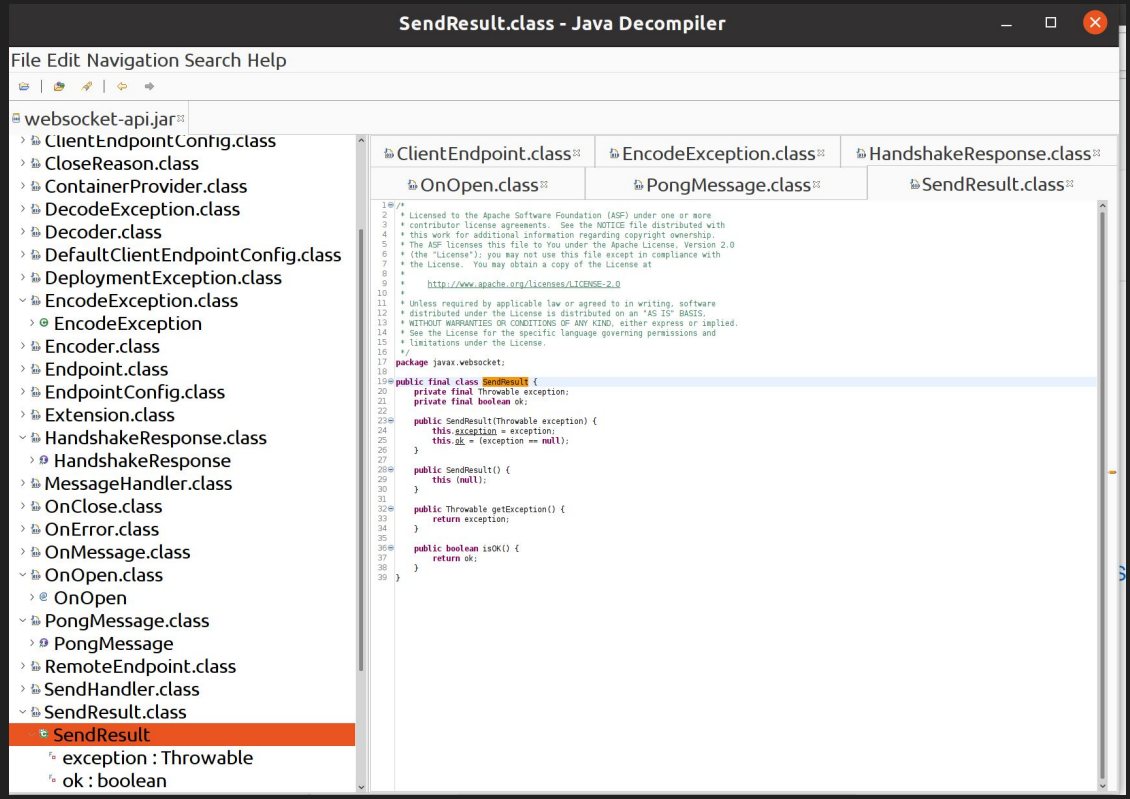
- .java files compile down to .class files
- Based on JVM bytecode for server side apps
  - The equivalent of .NET's MSIL

Java - JD-GUI

Reverse engineering tools for Java applications

- JD-GUI (https://github.com/java-decompiler/jd-gui)
- `brew install jd-gui` on MacOS
- Install from github releases on Linux
- Requires JDK 1.8 specifically
- Has sufficient decompiler output
- Can output all java files in a jar

JD-GUI Screenshot



Java - Fernflower

Fernflower is the JetBrains Java Decompiler

- Comes bundled with IntelliJ
- Can be run from the command line directly
- Has much clearer output than JD-GUI
- No UI, outputs .java files

What about other JVM Languages?

JD-GUI Scala:

```
1   import scala.Predef$;
2
3   public final class Hello$ {
4     public static final Hello$ MODULE$;
5
6     public void main(String[] args) {
7       Predef$.MODULE$.println("Hello, world");
8     }
9
10    private Hello$() {
11      MODULE$ = this;
12    }
13  }
14
```

Dotnet

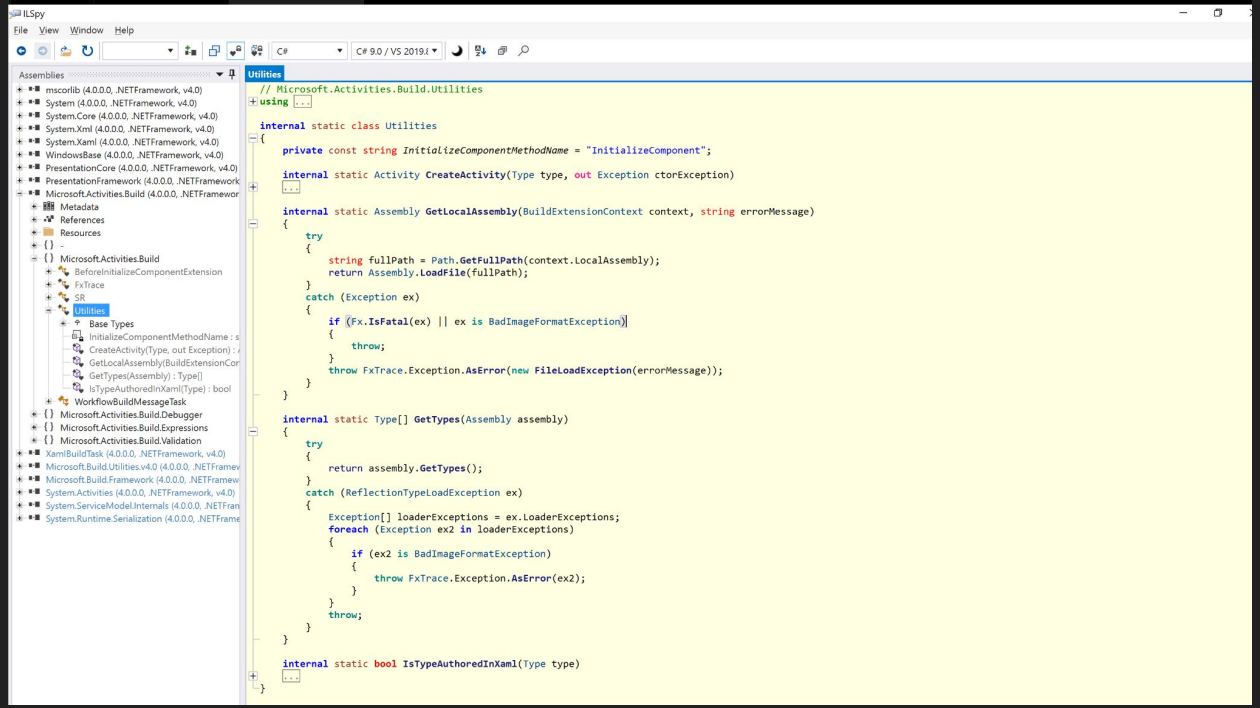
- Compiles down to MSIL (Microsoft Intermediate Language)
  - The .NET equivalent of JVM Bytecode
- This runs on the .NET CLR (Common language runtime)
- Source files are .cs files which compile to exe or dll
  - DLL’s more common for web apps

Dotnet - ILSpy

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

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

ILSpy Screenshot

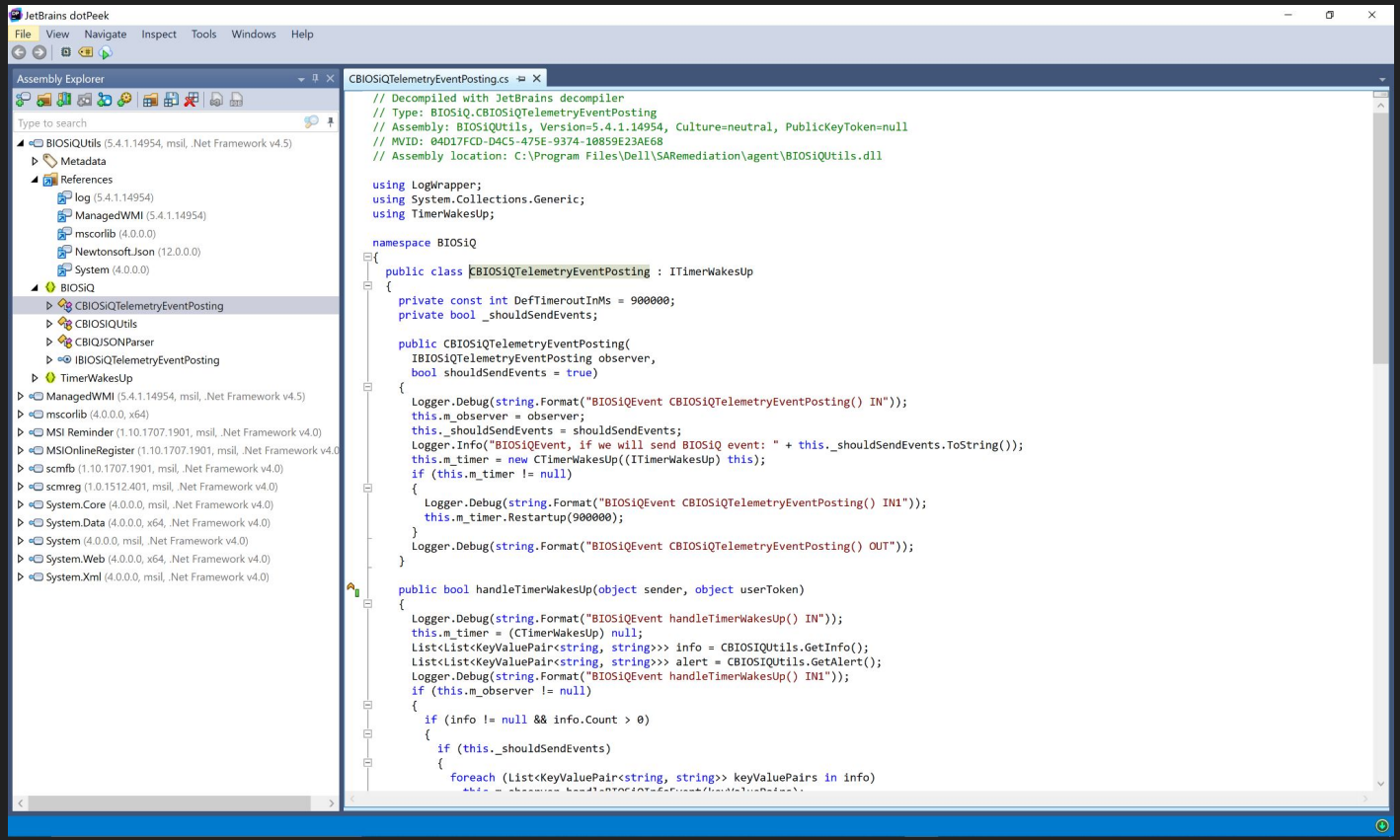


Dotnet - dotPeek

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

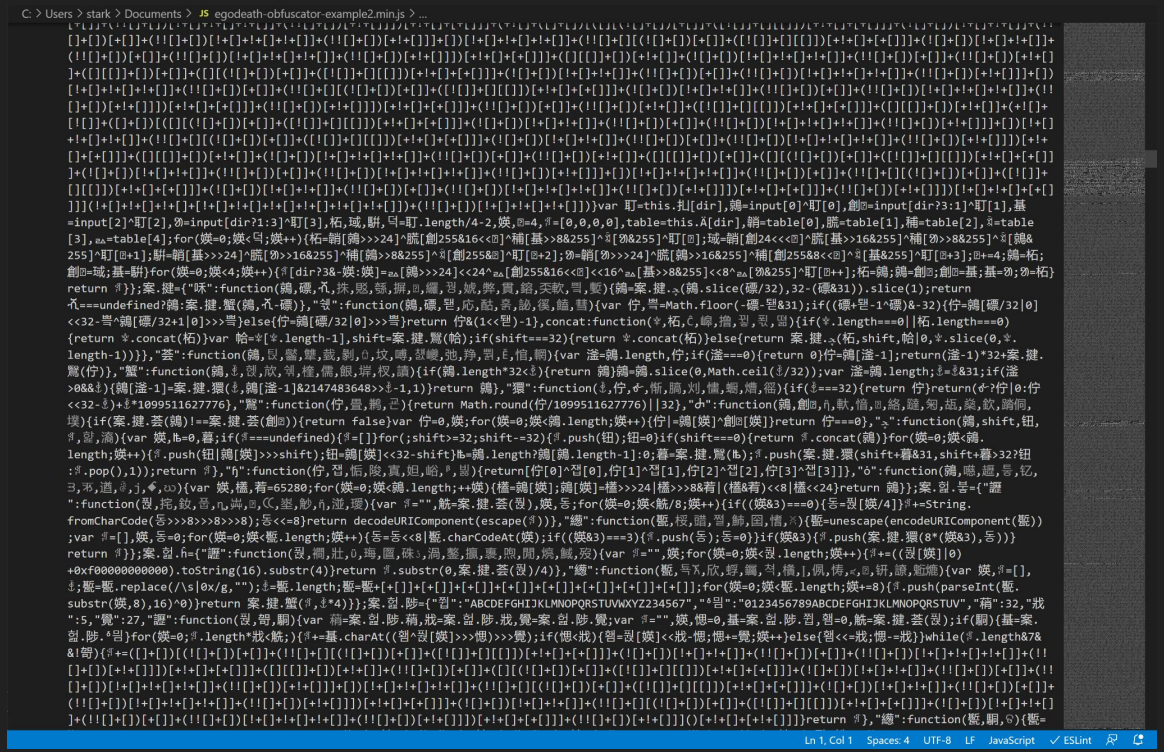


JavaScript

Not compiled down to bytecode / binary (uses JIT compilation for machine code instructions)

- Can be “minified” or “obfuscated” which makes JS difficult to read/comprehend
- Best tool to handle difficult to read JavaScript is js-beautify
- `npm i -g js-beautify`
- Runs from CLI

Example of Obfuscated JS



Anti-reverse engineering techniques

Obfuscation!

- Dotfuscator - dotnet
- Proguard - Java

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

Thank you!

Questions?

Contact:

https://twitter.com/nstarke

In depth presentation on dotnet / java reverse engineering coming later this fall at IADNUG and CIJUG - stay tuned!

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