Approaches to Reverse Engineering Java Applications

A survey of tools and techniques

Agenda

- Goals of Server-side application reverse engineering
- Server-side Java
- Client-side Java
- Modifying compiled JVM 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 Jasper/Jasmin for Java

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

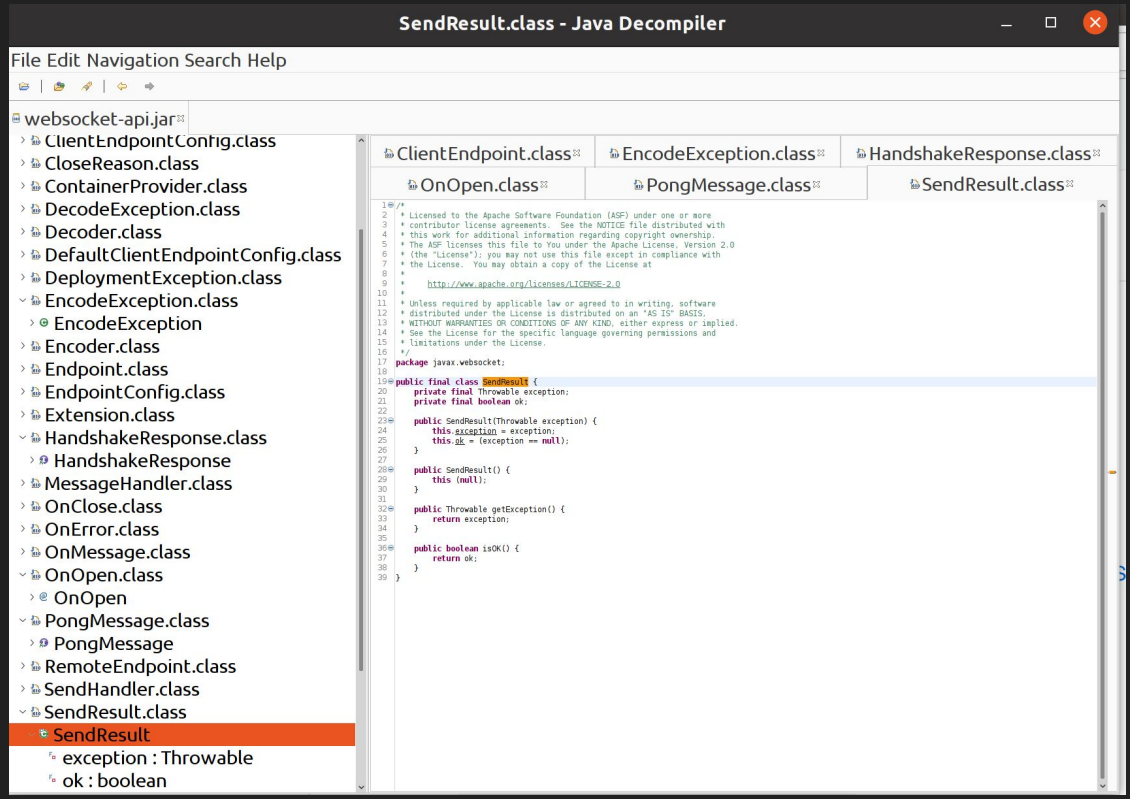
- .java files compile down to .class files
- Based on JVM bytecode for server side apps
  - The equivalent of .NET's MSIL

Server-side Java - JD-GUI

Reverse engineering tools for Server-side 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

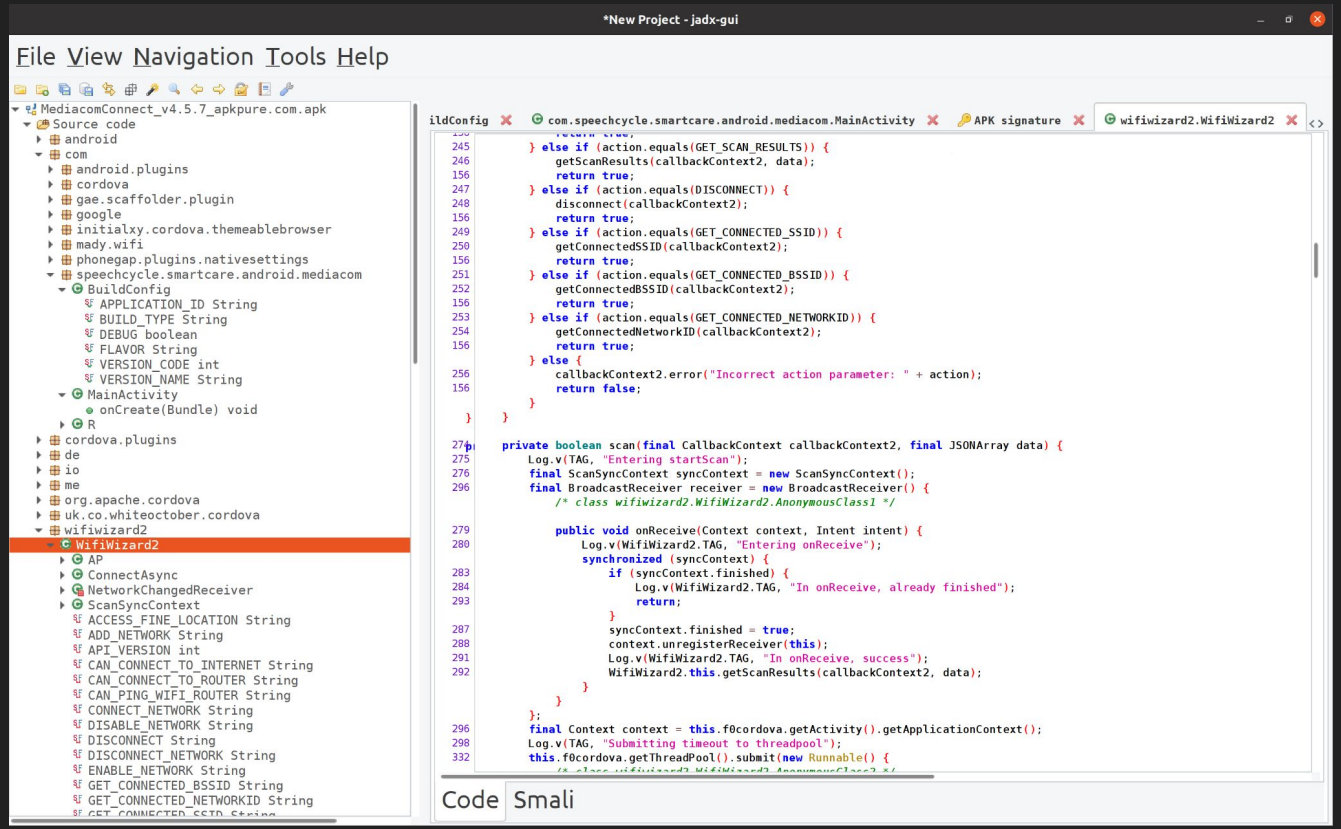
- Comes bundled with IntelliJ
- Can be run from the command line directly
- Has much clearer output than JD-GUI
- No UI, outputs .java files

Client-side Java

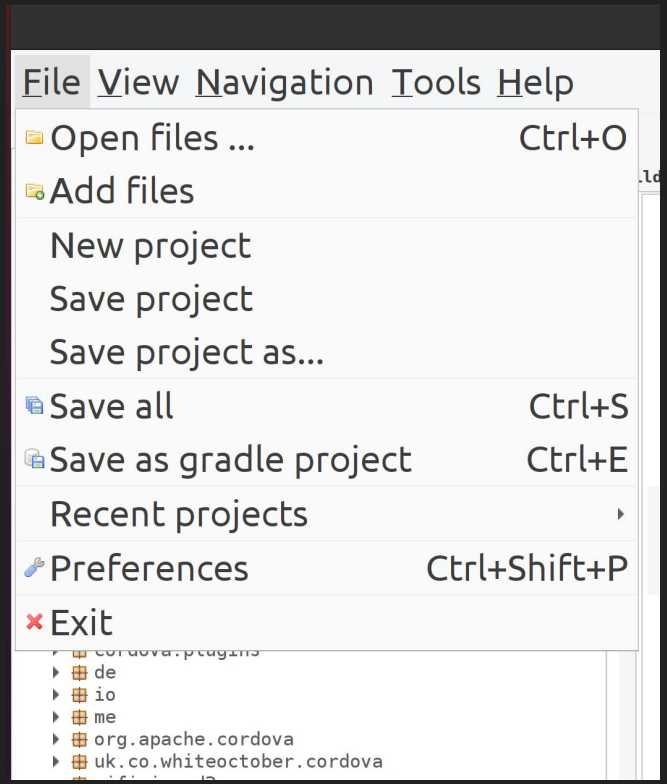
Jadx-gui - https://github.com/skylot/jadx

- A lot like JD-GUI
- Does all the manual work of extracting the APK then disassembling/decompiling the SMALI bytecode into Java classes
- Extracts the static content (res/) from the APK and presents it in a tree view

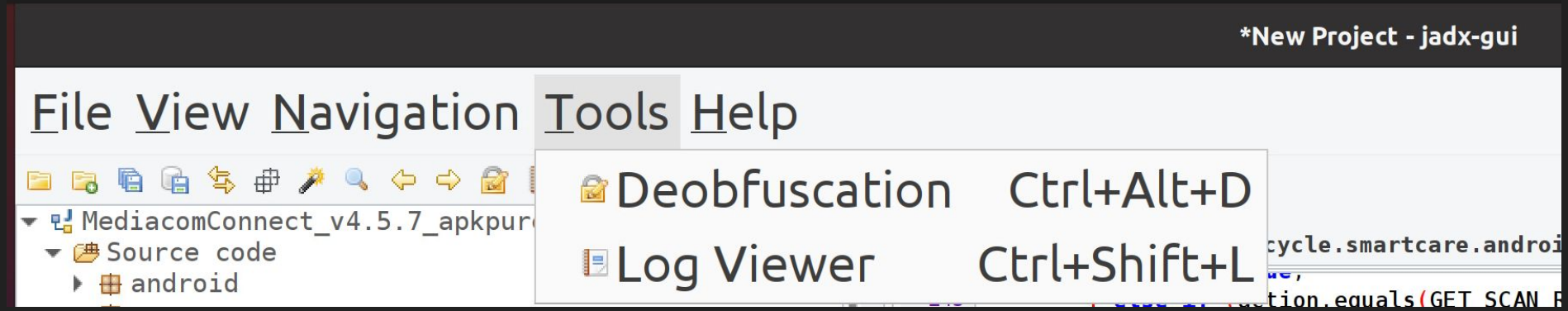
Jadx-gui screenshot



Jadx-gui file options screenshot



Jadx-gui tools options screenshot



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
```

Modifying JVM Bytecode without Source Code

The next few slides will focus on techniques for modifying JVM 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 JVM 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 Java 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 class file
6) Reassemble JAR / Drop on file system cache.

Java Disassembler / Assembler Duo

Java Class Disassembler: Jasper - https://github.com/kohsuke/jasper

Java Class Assembler: Jasmin - https://github.com/davidar/jasmin

- These two tools are built to work with each other.
- Jasmin will not work with “javap -c”!
- Both tools were built between 2000-2004
- Modifications to source are necessary for both to compile with modern Java tooling.
- Jasper works with maven, Jasmin works with ant.

How to mitigate this threat

- Jar signing (via `jarsigner` tool)
- Read only file system for executable code

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

Anti-reverse engineering techniques

Obfuscation!

- 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

Going Further

- Managed Code Rootkits (Book): https://www.amazon.com/Managed-Code-Rootkits-Hooking-Environments/dp/1597495743
- Covert Java (Book): https://www.amazon.com/dp/0672326388

Thank you!

Questions?

Contact:

https://twitter.com/nstarke

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

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