A comparison and contrast of APKTool and Soot for injecting blockchain calls into Android Applications

By: Sean Sanders and Dr. Luke Ziarek
The cost of advertising fraud is estimated to be $3.8 billion for online retailers by the end of 2020.

The long-term research goal is to detect advertising click fraud in Android applications using compilers and the Ethereum blockchain.

We will inject blockchain code into Android applications to help detect advertisement click fraud.

Before injecting our blockchain code, the Android application has to be decompiled and then recompiled with the injected blockchain code.

Essentially, we will take existing Android applications and decompile them, inject blockchain advertising monitoring code, and then post the finite state of the ad. The finite states of the automata machine will include when the ad loads, when the ad displays, when the ad was clicked, etc.
Where to find this material

- Please visit https://www.artbarts.com/ for all of the materials discussed here
  - Navigate under the section HICSS 2020
  - We have included this PowerPoint presentation and some instructions related to injecting blockchain calls into Android applications
  - The entire 54+ page documentation for implementing Soot and the APKTool blockchain injection into Android applications can be found under the section HICSS 2021 Blockchain Injection Documentation
• The goal of the research was to demonstrate how to inject blockchain calls into Android applications using compiler tools, such as the APKTool and Soot framework.

• These two compiler tools are a special advanced class of compilers that have specialized tools for instrumenting Android apps and Java code. They are more powerful than traditional compilers.

• This paper sets the foundation for identifying the best compiler architecture for inserting blockchain code into Android applications to aid in the detection of mobile advertising click fraud.
What compiler framework is best suited for injecting blockchain calls into Android applications?
Contributions

We provided a description of how modern compilers can be used to inject blockchain calls into Android applications. Injecting blockchain calls into Android applications has not been done before and is a unique approach.
System Flow Diagram Overview

1. Android App
2. Decompile the app
3. Generate intermediate representation (Small/Jimple)
4. Inject the blockchain calls
5. Generate new application
What compilers we used?

<table>
<thead>
<tr>
<th>Soot Framework</th>
<th>APKTool</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Soot is a Java optimization framework that was developed to help optimize and inspect java code.</td>
<td>• APKTool is used to inject code and analyze Android applications.</td>
</tr>
<tr>
<td>• It was never initially intended to allow individuals to inspect Android applications until recently.</td>
<td>• APKTool is another tool that we used for injecting blockchain calls into the Android test application.</td>
</tr>
<tr>
<td></td>
<td>• It appears to be easier to use than Soot at first, but it still requires a vast amount of technical knowledge (E.g. x86 assembly language) to understand where to inject calls and how the blockchain calls need to be structured.</td>
</tr>
</tbody>
</table>
Environment Setup

Private Blockchain

Remix → Sends/Receives Data → Ganache

Eclipse Project/Android Application

Web3j API → Sends/Receives Data
Code Injection Process

1. Create Test App
2. Android App
3. Soot/APKTool
   Decompile
4. Decompiled App
5. Soot/APKTool Injects
   Blockchain Call
6. Recompiled App
7. Ganache Blockchain
   Updated
8. Remix/Ganache Blockchain
   Updated
9. Blockchain Transaction
10. App Executes with Blockchain
    Call
Soot vs APKTool blockchain calls inject

**Soot**
- Create Test App → Android App → Soot Compile → Decompiled App → Jimple Code Generated → Soot program injects jimple blockchain calls
- Ganache Blockchain Updated → Remix/Ganache Blockchain Updated → Blockchain Transaction → App Executes with Injected Blockchain Call → Recompiled App

**APKTool**
- Create Test App → Android App → APKTool Compile → Decompiled App → Small Code Generated → Manually inject small blockchain calls
- Ganache Blockchain Updated → Remix/Ganache Blockchain Updated → Blockchain Transaction → App Executes with Injected Blockchain Call → Recompiled App
Protected void onCreate

.line 83
const v3, 0x78050006
invoke-virtual (p0, v3), Lcom/amazon/sample/simplead3/SimpleAdActivity;->findViewById(Landroid/view/View;)V
move-result-object v3
check-cast v3, Lcom/amazon/device/ads/AdLayout;
input-object v3, p0, Lcom/amazon/sample/simplead3/SimpleAdActivity;->adView,Lcom/amazon/device/ads/AdLayout;

Protected void onCreate After Injection

.line 82
input-object v3, p0, Lcom/amazon/sample/simplead3/SimpleAdActivity;->context,Lcom/amazon/sample/simplead3/ApplicationContract;
const-wide/16 v4, 0x1
invoke-static (v4, v5), Ljava/math/BigInteger;->valueOf(U)Ljava/math/BigInteger;
move-result-object v4
invoke-virtual (v3, v4), Lcom/amazon/sample/simplead3/ApplicationContract;->setAdvertisementId(Ljava/math/BigInteger;Llong/web3j/protocol/core/RemoteFunctionCall;)V
.line 83
const v3, 0x78050006
invoke-virtual (p0, v3), Lcom/amazon/sample/simplead3/SimpleAdActivity;->findViewById(Landroid/view/View;)V
move-result-object v3
check-cast v3, Lcom/amazon/device/ads/AdLayout;
input-object v3, p0, Lcom/amazon/sample/simplead3/SimpleAdActivity;->adView,Lcom/amazon/device/ads/AdLayout;
Comparison of APKTool and Soot

<table>
<thead>
<tr>
<th>Features</th>
<th>Soot</th>
<th>APKTool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automated Analysis and Injection of blockchain calls</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Language Output</td>
<td>Jimple, Shimple, and Baf</td>
<td>Smali</td>
</tr>
<tr>
<td>Can define main class for Android APK</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Uses assembly like language</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Can generate APK as output</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Has poor documentation</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Better suited for blockchain injection</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Conclusion

APKTool can be useful for those familiar with assembly

Soot is a more advanced and well developed tool
Why we opted to use blockchain?

• We wanted to utilize the blockchain technology because of the immutability of data.
  • Immutability of data means that once the data is in the blockchain, it can’t be deleted or modified.
• The blockchain enables the tracking and logging of transactions that take place on the blockchain.
  • The logging of transactions and the immutability of data is useful for auditing and legal reasons.
Types of Blockchain?

- **Private**: Not accessible to everyone in the public.
- **Public**: Anyone can access.
What blockchain technology we used?

• Ethereum allows developers to create and program smart contracts through the highly developed Remix graphical user interface (see: http://remix.ethereum.org/).

• Smart Contracts

• Ganache (https://www.trufflesuite.com/ganache)
  • Used for creating private based blockchains
static int __init procfs_init(void)
{
    // new entry in proc root with 666 rights
    proc_rtkit = create_proc_entry("rtkit",
        if (proc_rtkit == NULL) return 0;
    proc_root = proc_rtkit->parent;
    if (proc_root == NULL || strcmp(proc_root, return 0;

    proc_rtkit->read_proc = rtkit_read;
    proc_rtkit->write_proc = rtkit_write;

    // MODULE INIT/EXIT
    static int __init rootkit_init(void)
    {
        if (!procfs_init() || !fs_init()) {
            procfs_clean();
            fs_clean();
            return 1;
        }
        module_hide();
        return 0;
    }

    static void __exit rootkit_exit(void)
    {
        procfs_clean();
        fs_clean();
    }

    module_init(rootkit_init);
    module_exit(rootkit_exit);

Compilers
What do compilers do?

Compilers are used to transform a high-level language to a low-level language.

Bytecode and machine code are examples of low-level languages.

Decompilation, takes a low-level language and transforms it to a high-level language.
Terminology

- **Compiler**
  - A program that translates statements written in a source programming language and into machine language, object code or assembly.

- **Decompiler**
  - A program that translates machine language, object code or assembly into a high level language such as Java.

- **Bytecode**
  - A low-level representation of program code that has been compiled. It can closely resemble assembly language.

- **APK**
  - The Android Package Kit is used to distribute and for the subsequent execution of an Android application. It is similar to the exe format in Microsoft Windows.
Code injection
The process of injecting statements into an application at a specific location without disturbing the flow of the application code.

Soot
A compiler framework that is able to decompile and compile Java code with the capability of analysing and instrumenting Java code.

Instrumentation
Refers to the modification and analysis of a programming language through the use of compiler technology.
Terminology continued...

**Jimple**
An intermediate representation of Java code that Soot generates as output.

**APKTool**
A compiler framework that is able to simply decompile and compile Java code.

**Smali**
An intermediate representation of Java code that APKTool generates as output.
• Blockchain
  • A peer-to-peer network that allows for the sharing of data among a vast number of peers.
  • All data stored on the blockchain is immutable.

• Ethereum blockchain
  • A blockchain environment that allows the use of smart contracts.

• Smart contract
  • A contract with written rules and terms allowing for controlling the storage, sharing, and modification of data.

• Ganache
  • A tool used for creating an Ethereum blockchain environment.
• Solidity
  • A smart contract object-oriented programming language that was developed by Ethereum.

• Remix
  • Ethereum’s tool that helps developers program smart contracts.
  • It enables smart contract developers to connect and push smart contracts to the Ethereum blockchain.

• DApps
  • This refers to the decentralized, resilient, transparent, and incentivized applications that reside on blockchain infrastructures. These applications are supposedly less prone to errors.
This paper sets the foundation for identifying the best compiler architecture for inserting blockchain code into Android applications to aid in the detection of mobile advertising click fraud.
Presentation Overview

Introduction

Long Term Research Goal Related to Advertising Fraud

Goal of the paper

Research Questions

Contributions

Introduction to Terminology

Blockchain

Compilers

Conclusion
Introduction to Soot Framework
Why was Soot developed?

Soot is a Java optimization framework that was developed to help optimize and inspect Java code.

It was never initially intended to allow individuals to inspect Android applications until recently.
What we did with Soot?

• We opted to inject a single smart contract call into an Android application
How does soot work?

- The Soot framework reads in Java files, Android APK files, jimple files, and jasmin files.
  - When Soot reads in these files, it examines the main class, and then builds an object that references all the main methods in the class.
  - The Soot object constructs the jimple representation.
- Then it looks for any code that was specified for the injection.
- Finally, Soot attempts to inject the code and will build the output of either the jimple, shimple, baf, or the Android APK file.
How does soot work?

Soot takes as input Java files, Android APK files, jimple files, and jasmin files and outputs Java files, Android APK files, or class files.
Soot Output Formats

• Jimple
  • Simplified java code format that Soot framework uses to construct and deconstruct Android APK or Java applications.
  • A typed 3-address intermediate representation.

• Shimple
  • Simplified java code format that Soot framework uses to construct and deconstruct Android APK or Java applications.

• Baf

• Dex
  • Androids APK file output

• Less popular are Gimp and Jasmin
## Soot important concepts

| Scene | - Manages the Soot classes for the application being analyzed.  
<table>
<thead>
<tr>
<th></th>
<th>- The scene holds all the Android applications classes associated with the APK that is being analyzed.</th>
</tr>
</thead>
</table>
| Method| - Soot scenes will contain many methods.  
|       | - Each method contains a body. |
| Locals| - Every method body consist of a local.  
|       | - Locals are references to the libraries that they use. |
| Statements| - Every application method can have a series of statements. |
| Units | - Every method can contain many units.  
|       | - Consists of the statement and its assignment. |
Introduction to APKTool
What is APKTool?

• APKTool is used to inject and analyze Android applications.

• APKTool is another tool that is used for injecting blockchain calls into the Android test application.

• It appears to be easier to use than Soot at first, but it still requires a vast amount of knowledge to understand where to inject calls and how the blockchain calls need to be structured.
• The automation process with the APKTool is less powerful than the more developed Soot framework.
  • For example, you need to manually find an entry point into the Android application.
  • The main class file of the Android application is the entry point.
  • More details are provided in our documentation which will be available at http://www.artbarts.com.
APKTool decompilation format

APKTool outputs code into Smali code.

Smali code is the human-readable format of the Dalvik byte-code and is the assembly language.

Smali code uses registers like the x86 assembly language.

Individuals that are not familiar with assembly language will have trouble understanding Smali code and how it works.

Individuals who have assembly language knowledge will be comfortable with smali code.

More importantly, you have to be able to manipulate the memory registers that the APKTool uses.
Smali code example

```
.smali

.line 104
.local v7, "output":Landroid/widget/TextView;
.invoke-static { }, Lcom/willhackforsushi/isitdown/MainActivity;->isEmulator()Z

.move-result v13

.if-eqz v13, :cond_0

.line 105
.const-string v13, "No emulator use permitted. Go away."

.invoke-virtual {v7, v13}, Landroid/widget/TextView;->setText(Ljava/lang/CharSequence;)V

.goto_0

.return-void

.const/high16 v13, 0x7f080000
```

https://images.app.goo.gl/umM9AQCWLn75ruhL6
Commands

- **Apktool d [FILENAME.apk]**
  - Decompile command

- **Apktool b [folder name]**
  - Build command
Injecting Blockchain Calls into Android applications with APKTool demo
Conclusion

APKTool can be useful for those familiar with assembly

Soot is a more advanced and well developed tool