

Smart Phones, Dumb Apps

OWASP DC

Thursday November 11th, 2010

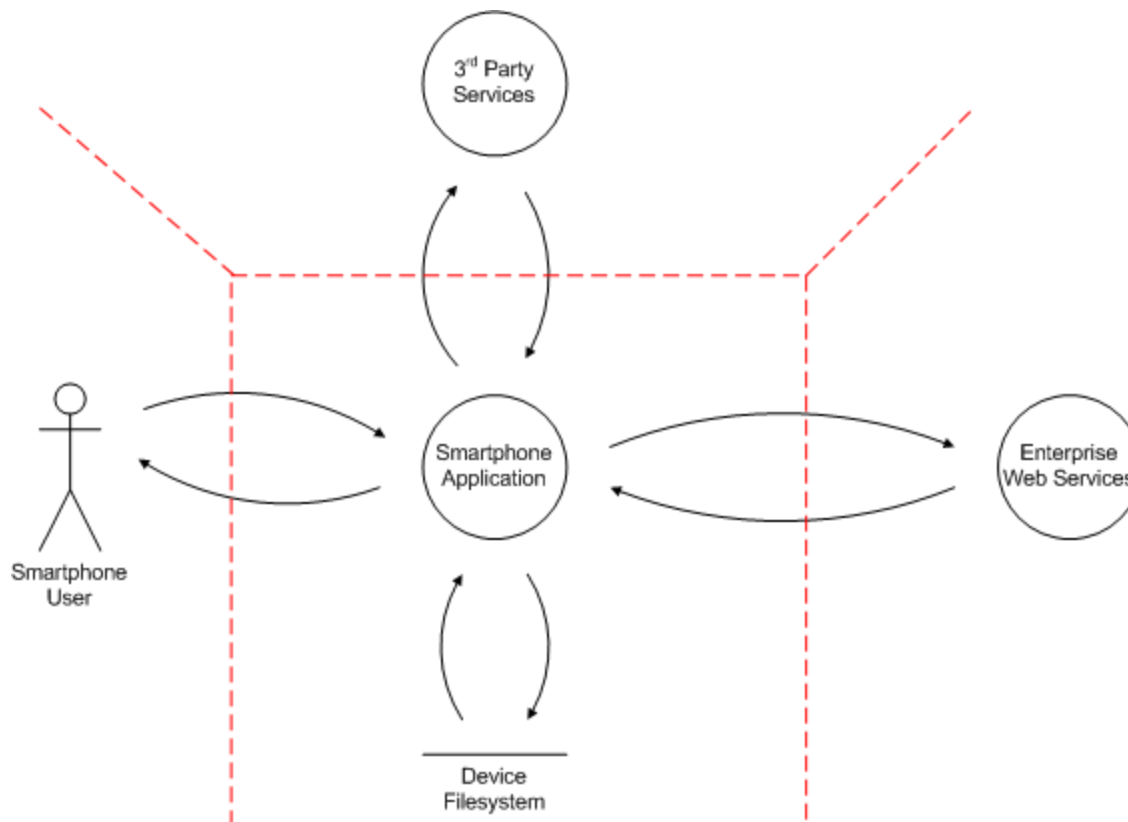
Agenda

- Generic Smartphone Threat Model
- Sample Application
- What an Attacker Sees (Android Edition)
- What About iPhones?
- Closing Thoughts
- Questions

Smart Phones, Dumb Apps

- Lots of media focus on device and platform security
 - *Important because successful attacks give tremendous attacker leverage*
- Most organizations:
 - *Accept realities of device and platform security*
 - *Concerned about the security of their custom applications*
 - *Concerned about sensitive data on the device because of their apps*
 - *Concerned about network-available resources that support their apps*
- Who has smartphone application deployed for customers?
- Who has had smartphone applications deployed without their knowledge?
 - **\$!%\$# marketing department...*

Generic Smartphone Threat Model



Some Assumptions for Developers

- Smartphone applications are essentially thick-client applications
 - *That people carry in their pockets*
 - *And drop in toilets*
 - *And put on eBay when the new iPhone comes out*
 - *And leave on airplanes*
 - *And so on...*
- Attackers will be able to access:
 - *Target user (victim) devices*
 - *Your application binaries*
- What else should you assume they know or will find out?

A Sample Application

- Attach to your brokerage account
- Pull stock quotes
- Make stock purchases

- (Apologies to anyone with any sense of UI design)

- This is intentionally nasty, but is it unrealistic?

So What Does a Bad Guy See? (Android Edition)

- Install the application onto a device
- Root the device
- Pull the application's APK file onto a workstation for analysis

- APK files are ZIP files
- They contain:
 - *AndroidManifest.xml*
 - *Other binary XML files in res/*
 - *classes.dex DEX binary code*

What's Up With My XML Files?



- Binary encoding
- Use axml2xml.pl to convert them to text

<http://code.google.com/p/android-random/downloads/detail?name=axml2xml.pl>

Do the Same Thing With the Rest of Them

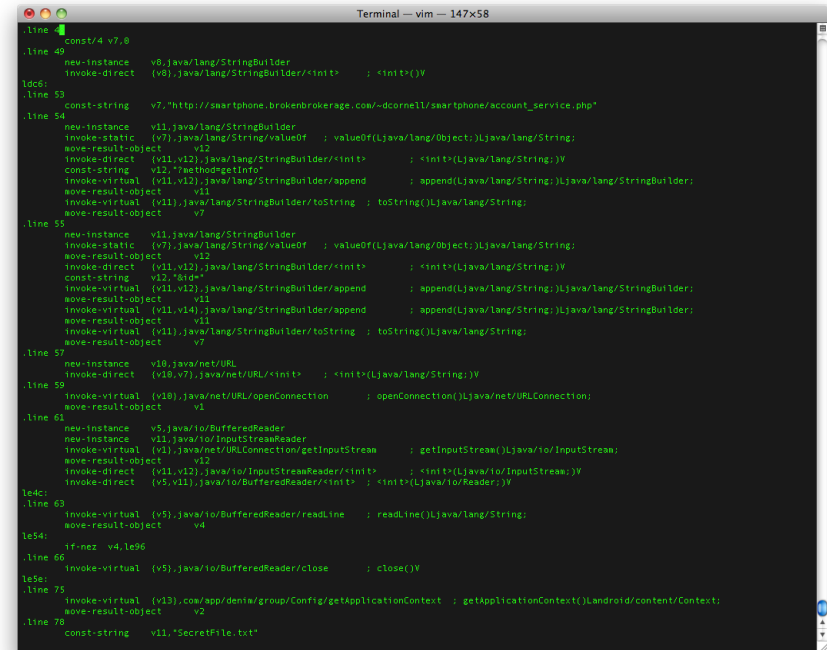
- Recurse through the res/ subdirectory
- UI layouts, other resources

What About the Code?

- All of it is stuffed in classes.dex
- Android phones use DEX rather than Java bytecodes
 - *Register-based virtual machine rather than stack-based virtual machine*
- Options:
 - *Look at DEX assembly via de-dexing*
 - *Convert to Java bytecode and then to Java source code*

Lots of Information

- Like the fun-fun world of Java disassembly and decompilation
 - *(We'll get to the DEX decompilation in a moment)*
- LOTS of information available



```
Terminal -- vim -- 147x58
Line 4
const/4 v7,0
Line 49
new-instance v8,java/lang/StringBuilder
invoke-direct (v8,java/lang/StringBuilder/<init> : <init>(J)V
ldc:
Line 53
const-string v7,"http://smartphone.brokenbrokerage.com/~dcornell/smartphone/account_service.php"
Line 54
new-instance v11,java/lang/StringBuilder
invoke-static (v7,java/lang/String/valueOf : valueOf(Ljava/lang/Object;)Ljava/lang/String;
move-result-object v12
invoke-direct (v11,v12,java/lang/StringBuilder/<init> : <init>(Ljava/lang/String;)V
const-string v12,"methodGetInfo"
invoke-virtual (v11,v12)java/lang/StringBuilder/append : append(Ljava/lang/String;)Ljava/lang/StringBuilder;
move-result-object v11
invoke-virtual (v11,java/lang/StringBuilder/toString : toString()Ljava/lang/String;
move-result-object v7
Line 55
new-instance v11,java/lang/StringBuilder
invoke-static (v7,java/lang/String/valueOf : valueOf(Ljava/lang/Object;)Ljava/lang/String;
move-result-object v12
invoke-direct (v11,v12,java/lang/StringBuilder/<init> : <init>(Ljava/lang/String;)V
const-string v12,"data"
invoke-virtual (v11,v12)java/lang/StringBuilder/append : append(Ljava/lang/String;)Ljava/lang/StringBuilder;
move-result-object v11
invoke-virtual (v11,v14)java/lang/StringBuilder/append : append(Ljava/lang/String;)Ljava/lang/StringBuilder;
move-result-object v11
invoke-virtual (v11,java/lang/StringBuilder/toString : toString()Ljava/lang/String;
move-result-object v7
Line 57
new-instance v10,java/net/URL
invoke-direct (v10,v7)java/net/URL/<init> : <init>(Ljava/lang/String;)V
Line 59
new-instance (v10,java/net/URL/openConnection : openConnection()Ljava/net/URLConnection;
move-result-object v1
Line 61
new-instance v5,java/io/BufferedReader
new-instance v11,java/io/InputStreamReader
invoke-virtual (v1)java/net/URLConnection/getInputStream : getInputStream()Ljava/io/InputStream;
move-result-object v12
invoke-direct (v11,v12)java/io/InputStreamReader/<init> : <init>(Ljava/io/InputStream;)V
invoke-direct (v5,v11)java/io/BufferedReader/<init> : <init>(Ljava/io/Reader;)V
ldc:
Line 63
invoke-virtual (v5,java/io/BufferedReader/readLine : readLine()Ljava/lang/String;
move-result-object v4
le54:
if-nez v4,le56
Line 66
invoke-virtual (v5,java/io/BufferedReader/close : close()V
le5e:
Line 75
invoke-virtual (v13,com/app/denim/group/Config/getApplicationContext : getApplicationContext()Landroid/content/Context;
move-result-object v2
Line 78
const-string v11,"SecretFile.txt"
```

But Can I Decompile to Java?

- Yes
- We
- Can

- Convert to Java bytecodes with dex2jar
 - <http://code.google.com/p/dex2jar/>
- Convert to Java source code with your favorite Java decompiler

DEX Assembly Versus Java Source Code

- De-DEXing works pretty reliably
- DEX assembly is easy to parse with grep
- DEX assembly is reasonably easy to manually analyze

- Java decompilation works most of the time
- Java source code can be tricky to parse with grep
- Java source code is very easy to manually analyze

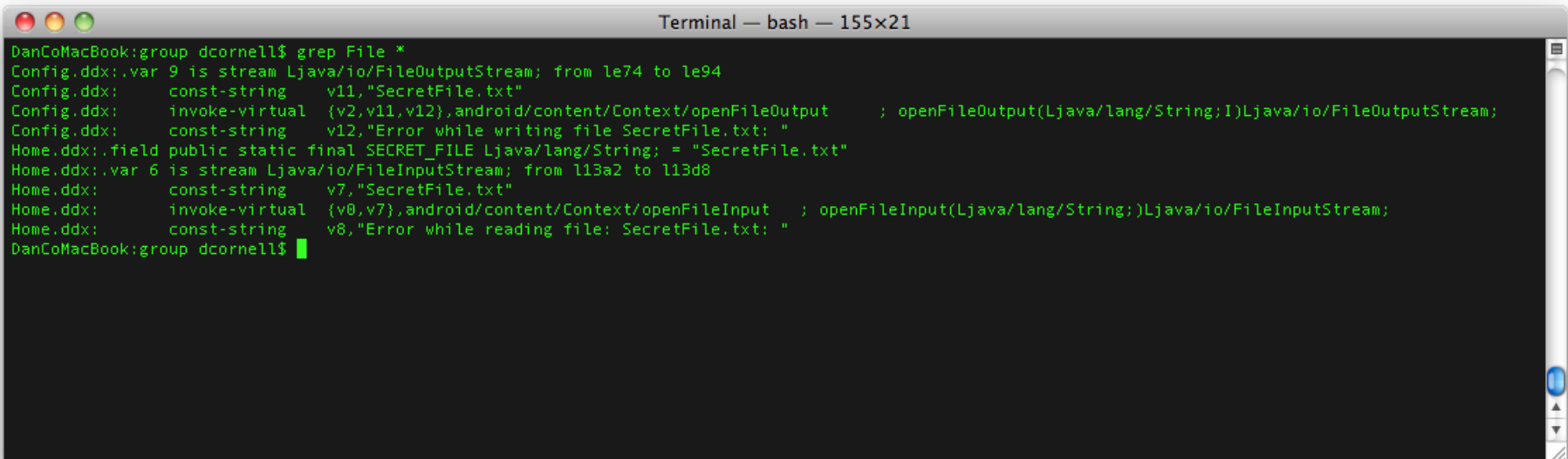
- Verdict:
 - *Do both!*
 - *Grep through DEX assembly*
 - *Analyze Java source*

So What Did We Learn?

- Look at the string constants
 - *URLs, hostnames, web paths*
- Look at the de-DEXed assembly
 - *Method calls*
 - *Data flow*
- Developers: BAD NEWS
 - *The bad guys have all your code*
 - *They might understand your app better than you*

Is There Sensitive Data On the Device?

- Look at the code
- Grep for “File”



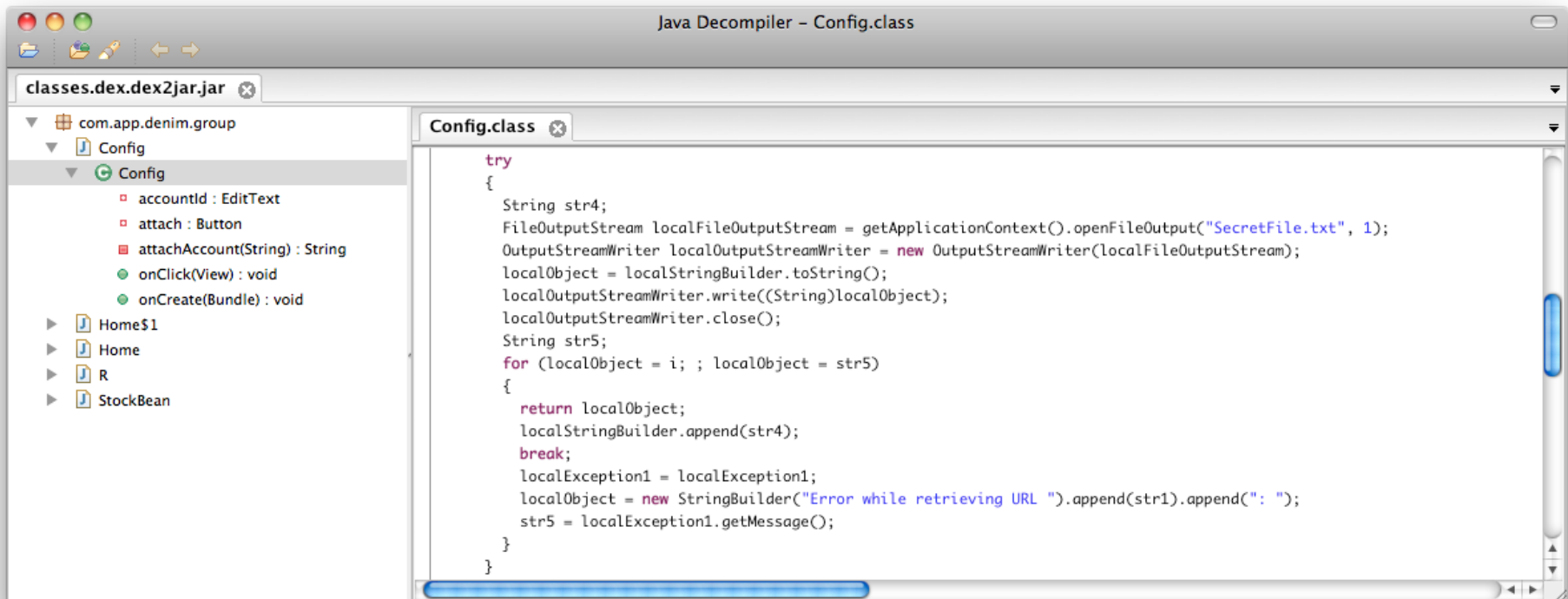
```
Terminal — bash — 155x21
DanCoMacBook:group dcornell$ grep File *
Config.ddx:.var 9 is stream Ljava/io/FileOutputStream; from 1e74 to 1e94
Config.ddx:  const-string    v11,"SecretFile.txt"
Config.ddx:  invoke-virtual  {v2,v11,v12},android/content/Context/openFileOutput    ; openFileOutput(Ljava/lang/String;I)Ljava/io/FileOutputStream;
Config.ddx:  const-string    v12,"Error while writing file SecretFile.txt: "
Home.ddx:.field public static final SECRET_FILE Ljava/lang/String; = "SecretFile.txt"
Home.ddx:.var 6 is stream Ljava/io/FileInputStream; from 113a2 to 113d8
Home.ddx:  const-string    v7,"SecretFile.txt"
Home.ddx:  invoke-virtual  {v0,v7},android/content/Context/openFileInput    ; openFileInput(Ljava/lang/String;)Ljava/io/FileInputStream;
Home.ddx:  const-string    v8,"Error while reading file: SecretFile.txt: "
DanCoMacBook:group dcornell$
```

What About Java Source Code?

- Get the source code with JD-Gui
 - <http://java.decompiler.free.fr/>

Look for Files With Bad Permissions

- Look for file open operations using
 - `Context.MODE_WORLD_READABLE`
 - (translates to “1”)



```
Java Decompiler - Config.class
classes.dex.dex2jar.jar
com.app.denim.group
  Config
    Config
      accountId : EditText
      attach : Button
      attachAccount(String) : String
      onClick(View) : void
      onCreate(Bundle) : void
    Home$1
    Home
    R
    StockBean

Config.class
try
{
    String str4;
    FileOutputStream localFileOutputStream = getApplicationContext().openFileOutput("SecretFile.txt", 1);
    OutputStreamWriter localOutputStreamWriter = new OutputStreamWriter(localFileOutputStream);
    localObject = localStringBuilder.toString();
    localOutputStreamWriter.write((String)localObject);
    localOutputStreamWriter.close();
    String str5;
    for (localObject = i; ; localObject = str5)
    {
        return localObject;
        localStringBuilder.append(str4);
        break;
        localException1 = localException1;
        localObject = new StringBuilder("Error while retrieving URL ").append(str1).append(": ");
        str5 = localException1.getMessage();
    }
}
```

Next: What Is On the Server-Side

- To access sensitive data on a device:
 - *Steal a device*
 - *Want more data?*
 - *Steal another device*
- To access sensitive data from web services
 - *Attack the web service*
- String constants for URLs, hostnames, paths
- Examples:
 - *3rd party web services*
 - *Enterprise web services*

So Now What?

- 3rd Party Web Services
 - *Is data being treated as untrusted?*
- Enterprise Web Services
 - *Did you know these were deployed?*

Web Services Example

- Trumped up example, but based on real life
- Given a web services endpoint, what will a bad guy do?

What Is Wrong With the Example Application?

- Sensitive data stored on the device
- Trusts data from 3rd party web services
- Exposes enterprise web services
- Enterprise web services vulnerable to XSS attacks
- And so on...

What About iPhones?

- Objective-C compiled to ARMv6 machine code
 - *Not as fun as Java compiled to DEX bytecode*
- Apps from iTunes Store
 - *Encrypted*
 - *Used to be “easy” (well, mechanical) to break encryption with a jailbroken phone and a debugger*
 - *Now trickier*
 - *But the default apps are not encrypted...*

Run “strings” on the Binary

- Web services endpoints: URLs, hostnames, paths
- Objective-C calling conventions:

```
[myThing doStuff a b c];
```

becomes

```
obj_msgsend(myThing, “doStuff:”, a, b, c);
```

Run “otool” on the Binary

- `otool -l <MyApp>`
 - *View the load commands*
 - *Segment info, encryption info, libraries in use*
- `otool -t -v <MyApp>`
 - *Disassemble the text segment to ARMv6 assembly*
 - *If run on an encrypted application you get garbage*
- And so on...

Decoding Files: Easy for iPhones Too

```
Terminal — vim — 90x60
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>CFBundleDevelopmentRegion</key>
  <string>English</string>
  <key>CFBundleExecutable</key>
  <string>Maps</string>
  <key>CFBundleIdentifier</key>
  <string>com.apple.Maps</string>
  <key>CFBundleInfoDictionaryVersion</key>
  <string>6.0</string>
  <key>CFBundlePackageType</key>
  <string>APPL</string>
  <key>CFBundleResourceSpecification</key>
  <string>ResourceRules.plist</string>
  <key>CFBundleSignature</key>
  <string>????</string>
  <key>CFBundleSupportedPlatforms</key>
  <array>
    <string>iPhoneOS</string>
  </array>
  <key>CFBundleURLTypes</key>
  <array>
    <dict>
      <key>CFBundleURLSchemes</key>
      <array>
        <string>maps</string>
      </array>
    </dict>
  </array>
  <key>CFBundleVersion</key>
  <string>1.0</string>
  <key>DTCompiler</key>
  <string>4.2</string>
  <key>DTPlatformName</key>
  <string>iPhoneOS</string>
  <key>DTPlatformVersion</key>
  <string>4.0 GM</string>
  <key>DTSDKName</key>
  <string>iPhoneOS4.0.internal</string>
  <key>DTXcode</key>
  <string>0323</string>
  <key>MinimumOSVersion</key>
  <string>4.0</string>
  <key>SBMatchingApplicationGenres</key>
  <array>
    <string>Navigation</string>
    <string>Travel</string>
    <string>Reference</string>
  </array>
  <key>SBUsesNetwork</key>
  <integer>3</integer>
  <key>UIDeviceFamily</key>
  <array>
    <integer>1</integer>
    <integer>2</integer>
  </array>
</plist>
```

- `plutil -convert xml1 Info.plist`
- Much nicer
- XPath: Look for:
`/plist/dict/array/dict[key='CFBundleURLSchemes']/array/string`
- Now you know the URL Schemes the app handles

Net Result for iPhone

- More obscure
 - *But does that mean more secure?*
- Can still retrieve a tremendous amount of information

So What Should Developers Do?

- Threat model your smartphone applications
 - *More complicated architectures -> more opportunities for problems*
- Watch what you store on the device
 - *May have PCI, HIPAA implications*
- Be careful consuming 3rd party services
 - *Who do you love? Who do you trust?*
- Be careful deploying enterprise web services
 - *Very attractive target for bad guys*
 - *Often deployed “under the radar”*

So What Should Security People Do?

- Find out about smartphone projects
 - *Not always done by your usual development teams*
 - *R&D, “Office of the CTO,” Marketing*
- Assess the security implications of smartphone applications
 - *What data is stored on the device?*
 - *What services are you consuming?*
 - *Are new enterprise services being deployed to support the application?*

Resources

- axml2xml.pl (Convert Android XML files to normal XML)
 - <http://code.google.com/p/android-random/downloads/detail?name=axml2xml.pl>
- Dedexer (Convert DEX bytecodes into DEX assembler)
 - <http://dedexer.sourceforge.net/>
- Dex2jar (Convert DEX bytecode in Java bytecode)
 - <http://code.google.com/p/dex2jar/>
- JD-GUI (Convert Java bytecode to Java source code)
 - <http://java.decompiler.free.fr/>
- otool (Get information about iPhone binaries)
 - <http://developer.apple.com/library/mac/#documentation/Darwin/Reference/ManPages/man1/otool.1.html>

Online

- Code/etc online:

www.smartphonesdumbapps.com

Questions?

Dan Cornell

dan@denimgroup.com

Twitter: [@danielcornell](https://twitter.com/danielcornell)

www.denimgroup.com

(210) 572-4400