

Smart Phones Dumb Apps

Dan Cornell

My Background

- Dan Cornell, founder and CTO of Denim Group
- Software developer by background (Java, .NET, etc)
- OWASP San Antonio, Global Membership Committee

- Denim Group
 - *Build software with special security, performance, reliability requirements*
 - *Help organizations deal with the risk associated with their software*
 - Code reviews and application assessments
 - SDLC consulting
 - Secure development training – instructor-led and [eLearning](#)

Agenda

- Generic Smartphone Threat Model
- Sample Application
- What an Attacker Sees (Android Edition)
- What About iPhones/iPads?
- Special Topic: Browser URL handling
- Closing Thoughts
- Questions

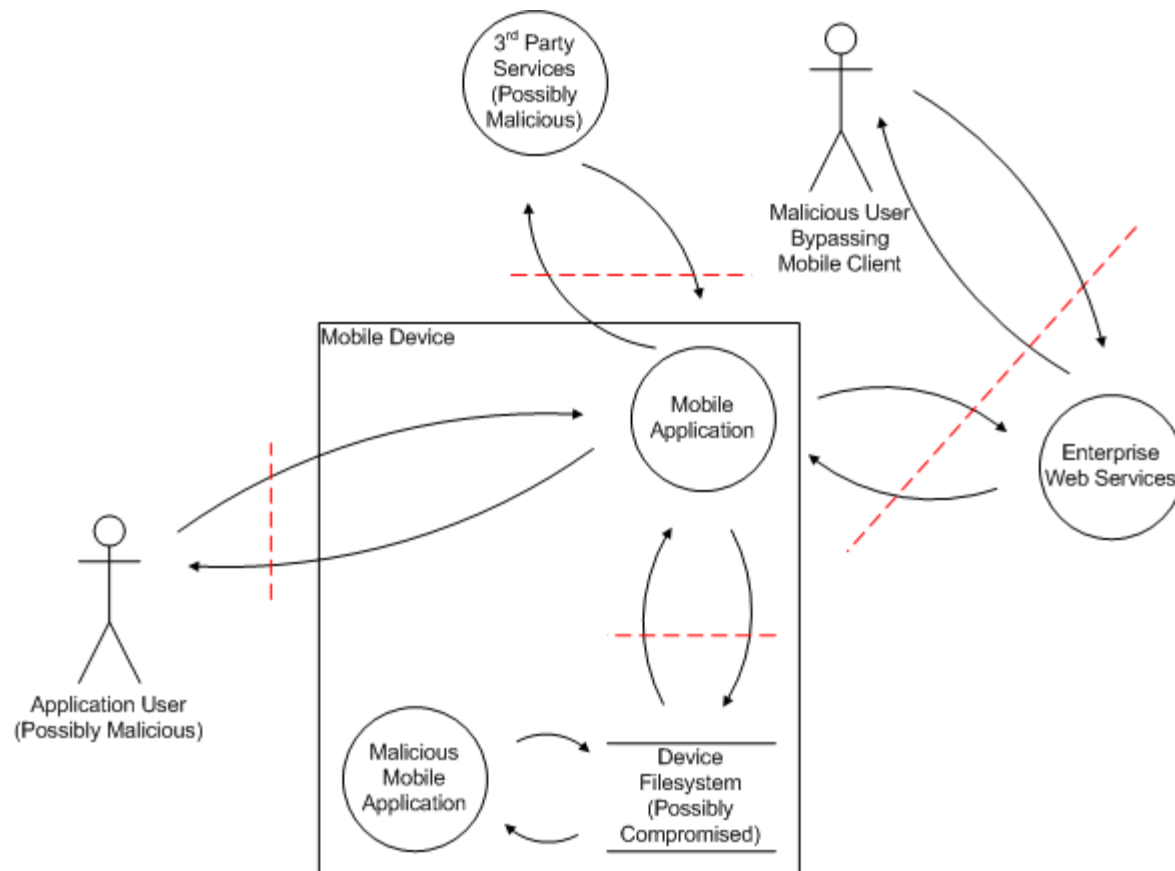
Tradeoffs: Value versus Risk

- Mobile applications can create tremendous value for organizations
 - *New classes of applications utilizing mobile capabilities: GPS, camera, etc*
 - *Innovating applications for employees and customers*
- Mobile devices and mobile applications can create tremendous risks
 - *Sensitive data inevitably stored on the device (email, contacts)*
 - *Connect to a lot of untrusted networks (carrier, WiFi)*
- Most developers are not trained to develop secure applications
 - *Fact of life, but slowing getting better*
- Most developers are new to creating mobile applications
 - *Different platforms have different security characteristics and capabilities*

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 Mobile Application 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

- Application on mobile device supported by enterprise and 3rd party web services

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

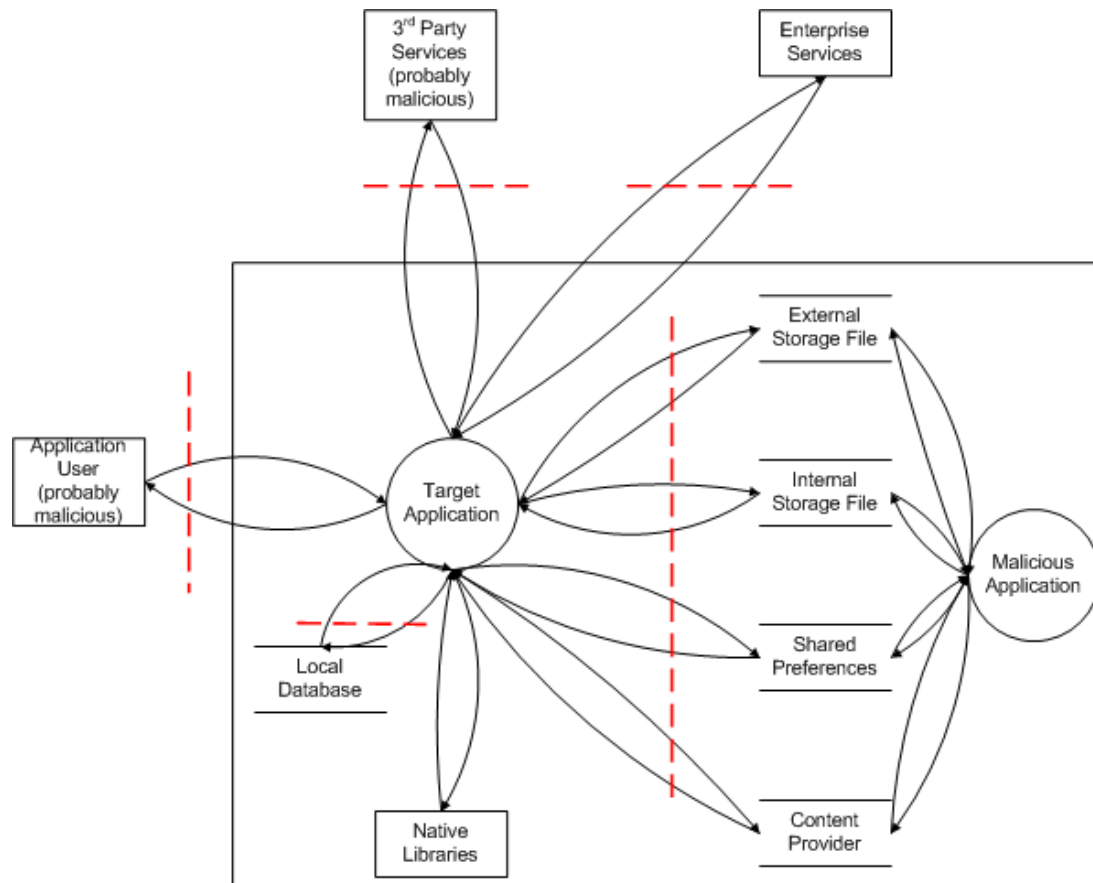
- This is intentionally nasty, but is it unrealistic?

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

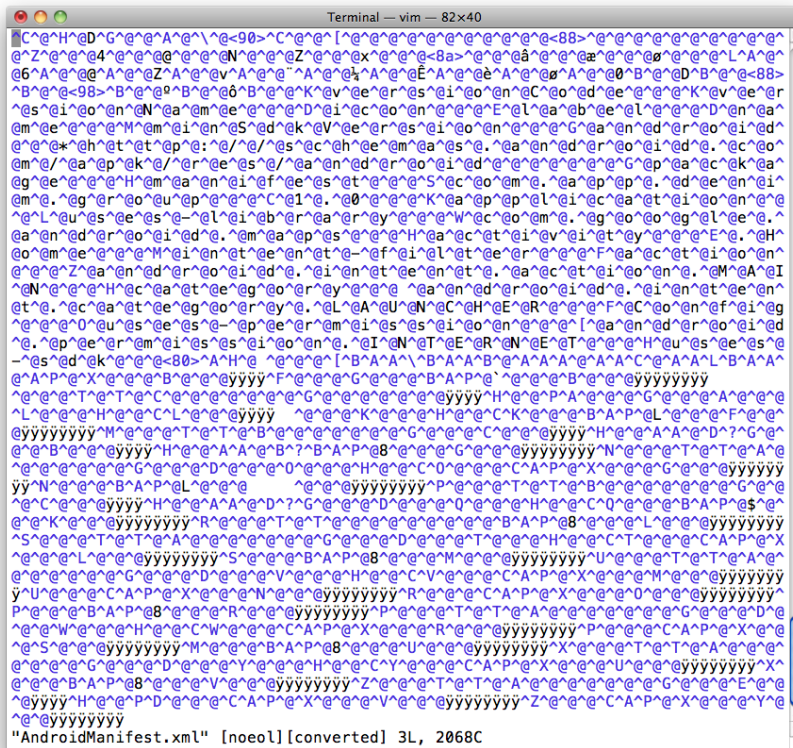
- Install the application onto a device
- Root the device
- Pull the application's APK file onto a workstation for analysis

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

Generic Android Application Threat Model



What's Up With My XML Files?



```
Terminal - vim - 82x40
[Content of the terminal window is garbled text, likely binary data being viewed in a text editor.]
"AndroidManifest.xml" [noeol][converted] 3L, 2068C
```

- 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

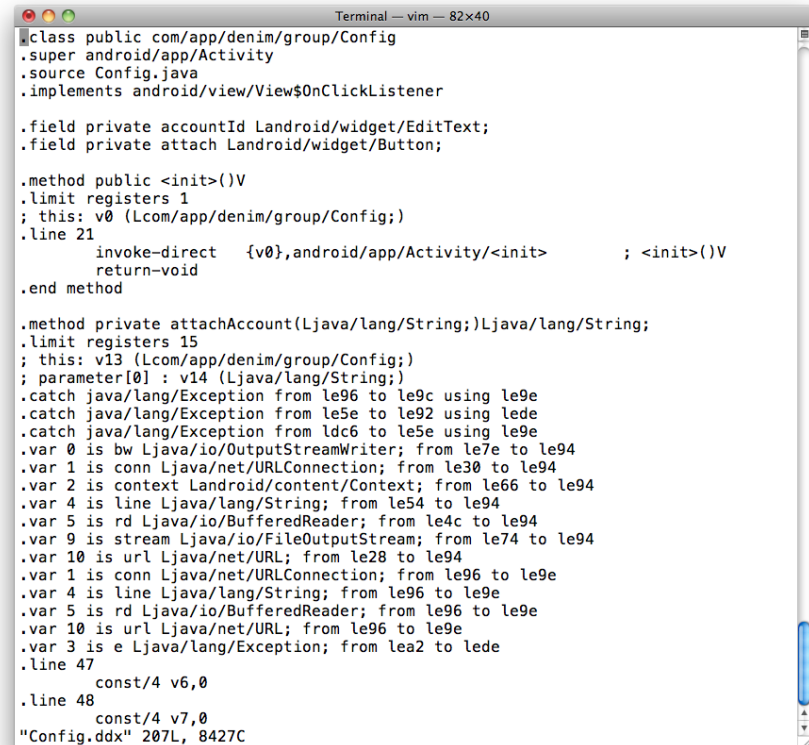
- Recurse through the res/ subdirectory
- UI layouts, other resources

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 — 82x40
class public com/app/denim/group/Config
.super android/app/Activity
.source Config.java
.implements android/view/View$OnClickListener

.field private accountId Landroid/widget/EditText;
.field private attach Landroid/widget/Button;

.method public <init>()V
.limit registers 1
; this: v0 (Lcom/app/denim/group/Config;)
.line 21
    invoke-direct    {v0},android/app/Activity/<init>    ; <init>()V
    return-void
.end method

.method private attachAccount(Ljava/lang/String;)Ljava/lang/String;
.limit registers 15
; this: v13 (Lcom/app/denim/group/Config;)
; parameter[0] : v14 (Ljava/lang/String;)
.catch java/lang/Exception from le96 to le9c using le9e
.catch java/lang/Exception from le5e to le92 using lede
.catch java/lang/Exception from ldc6 to le5e using le9e
.var 0 is bw Ljava/io/OutputStreamWriter; from le7e to le94
.var 1 is conn Ljava/net/URLConnection; from le30 to le94
.var 2 is context Landroid/content/Context; from le66 to le94
.var 4 is line Ljava/lang/String; from le54 to le94
.var 5 is rd Ljava/io/BufferedReader; from le4c to le94
.var 9 is stream Ljava/io/FileOutputStream; from le74 to le94
.var 10 is url Ljava/net/URL; from le28 to le94
.var 1 is conn Ljava/net/URLConnection; from le96 to le9e
.var 4 is line Ljava/lang/String; from le96 to le9e
.var 5 is rd Ljava/io/BufferedReader; from le96 to le9e
.var 10 is url Ljava/net/URL; from le96 to le9e
.var 3 is e Ljava/lang/Exception; from lea2 to lede
.line 47
    const/4 v6,0
.line 48
    const/4 v7,0
"Config.ddx" 207L, 8427C
```


But Can I Decompile to Java?

- Yes
- We
- Can

- Convert to Java bytecodes with dex2jar
 - <http://code.google.com/p/dex2jar/>
 - *(Now you can run static analysis tools like Findbugs)*

- Convert to Java source code with your favorite Java decompiler
 - *Everyone has a favorite Java decompiler, right?*

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

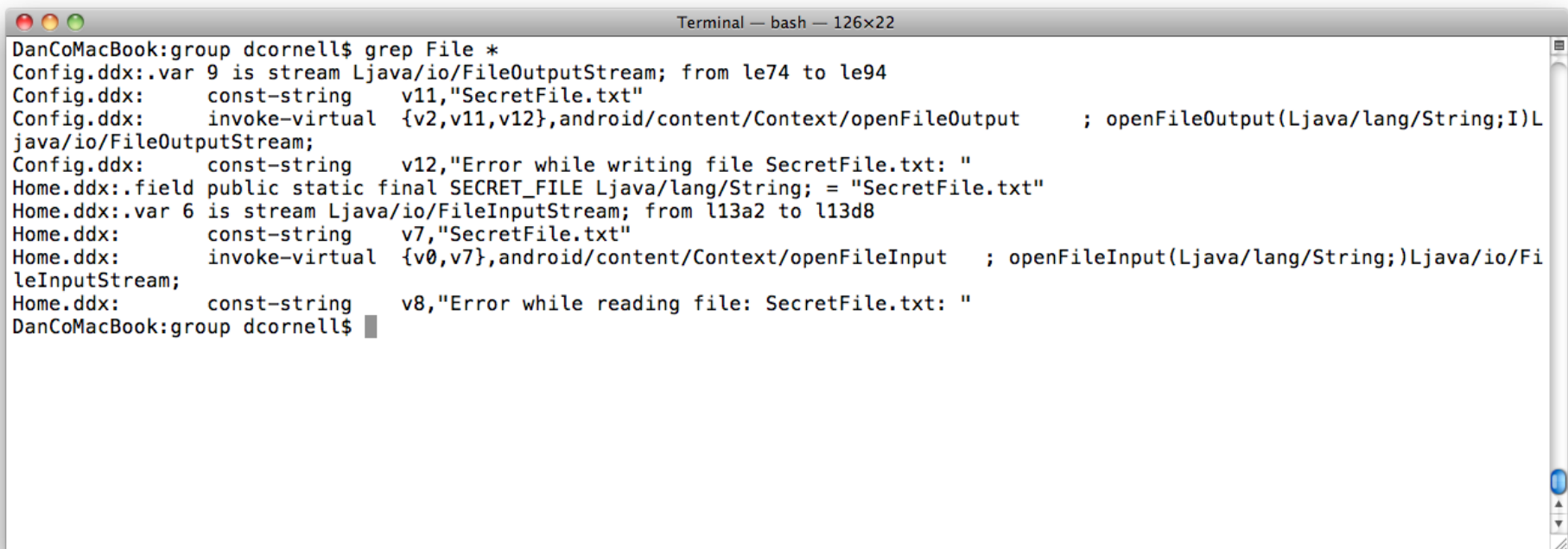
- Verdict:
 - *Do both!*
 - *Grep through DEX assembly to identify starting points for analysis*
 - *Analyze Java source in detail*

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*
 - *How much sensitive intellectual property do you want to embed in your mobile application now?*

Is There Sensitive Data On the Device?

- Look at the disassembled DEX code
- Grep for “File”



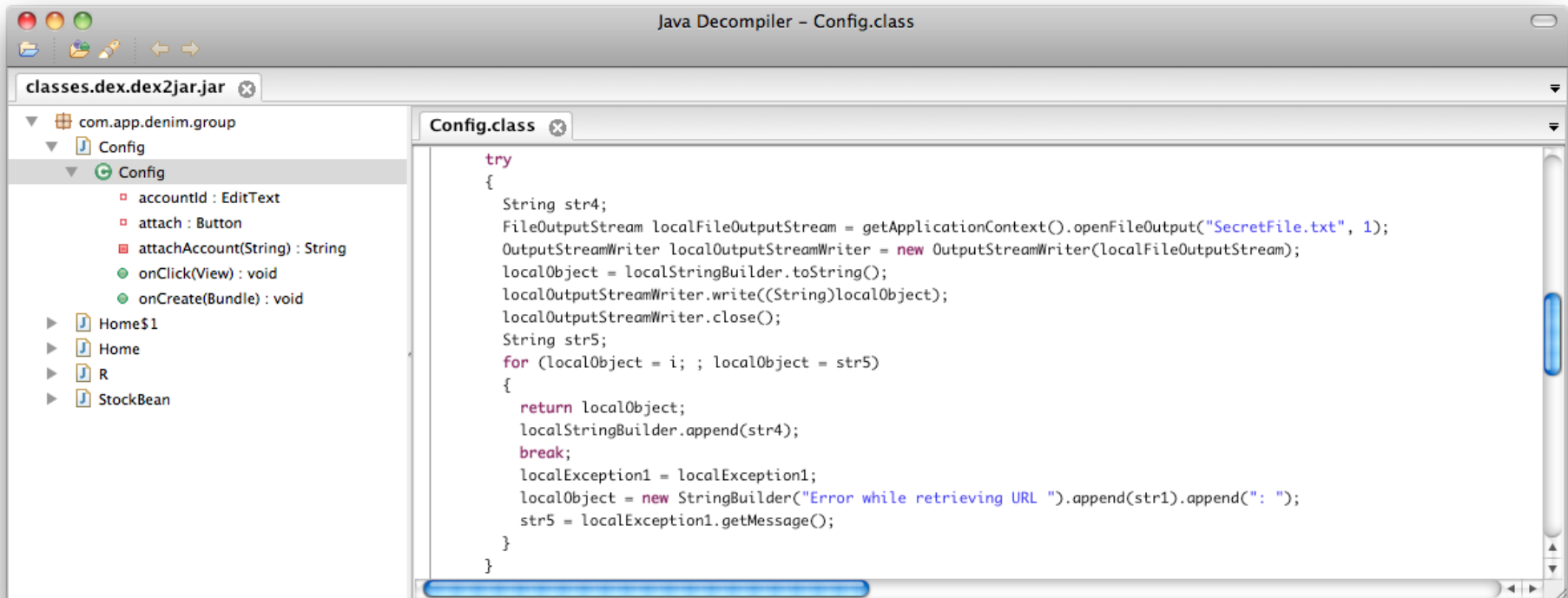
```
Terminal — bash — 126x22
DanCoMacBook:group dcornell$ grep File *
Config.ddx:.var 9 is stream Ljava/io/FileOutputStream; from le74 to le94
Config.ddx:    const-string    v11,"SecretFile.txt"
Config.ddx:    invoke-virtual  {v2,v11,v12},android/content/Context/openFileOutput    ; openFileOutput(Ljava/lang/String;I)L
java/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 l13a2 to l13d8
Home.ddx:    const-string    v7,"SecretFile.txt"
Home.ddx:    invoke-virtual  {v0,v7},android/content/Context/openFileInput    ; openFileInput(Ljava/lang/String;)Ljava/io/Fi
leInputStream;
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”)



```
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?*
 - *Google promised to “not be evil”*
 - For everyone else...
- Enterprise Web Services
 - *Did you know these were deployed?*
 - *Have these been tested for possible security flaws?*
 - *Stealing records en-masse is preferable to stealing them one-at-a-time*

Web Services Example

- Trumped up example, but based on real life
- Given a web services endpoint, what will a bad guy do?
- Sequence:
 - *Request a junk method “abcd”*
 - *Get a “No method ‘abcd’ available”*
 - *Request a method “<script>alert(‘hi’);</script>”*
 - *Hilarity ensues...*

What Is Wrong With the Example Application?

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

- This is a trumped-up example with concentrated vulnerabilities, but...

- All of these reflect real-world examples of vulnerabilities
 - *Public breaches*
 - *Application assessments*

What About iPhones/iPads?

- Objective-C compiled to ARMv6, ARMv7 machine code
 - *Not as fun (easy) as Java compiled to DEX bytecode*
 - *But ... subject to buffer overflows, memory handling issues, other native code fun*
- Apps from iTunes Store
 - *Encrypted*
 - *Used to be “easy” (well, mechanical) to break encryption with a jailbroken phone and a debugger*
 - *Now trickier (but likely not insurmountable)*
 - *And the default apps are not encrypted...*

Run “strings” on the Binary

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

```
[myThing doStuff:a second:b third:c];
```

becomes

```
obj_msgsend(myThing, “doStuff:second:third:”, 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...

Net Result for iPhone/iPad

- More obscure
 - *But does that mean more secure?*
- Can still retrieve a tremendous amount of information
- Can still observe a running application
- “Security” based on obscurity is not durable

Mobile Browser Content Handling

- Many mobile platforms allow you to designate applications to handle content found in web pages
 - *By URI protocol*
 - *By content type*
- Provide a “premium” experience for users who have the target app installed
- Examples:
 - *[tel://](#) URLs initiating phone calls*
 - *[maps://](#) URLs to display maps*

Decoding plist Files

- `plutil -convert xml1 Info.plist`
- Much nicer

```
Terminal -- bash -- 82x40
?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>
  "Info.plist" [readonly] 60L, 1500C
```

iOS URL Handlers

- XPath: Look for:
`/plist/dict/array/dict[key='CFBundleURLSchemes']/array/string`
- Now you know the URL Schemes the app handles
- SANS blog post on this issue in iOS:
 - http://software-security.sans.org/blog/2010/11/08/insecure-handling-url-schemes-apples-ios/?utm_source%253Drss%2526utm_medium%253Drss%2526utm_campaign%253Dinsecure-handling-url-schemes-apples-ios
 - Too long to type? <http://bit.ly/ezqdK9>

Android Intents

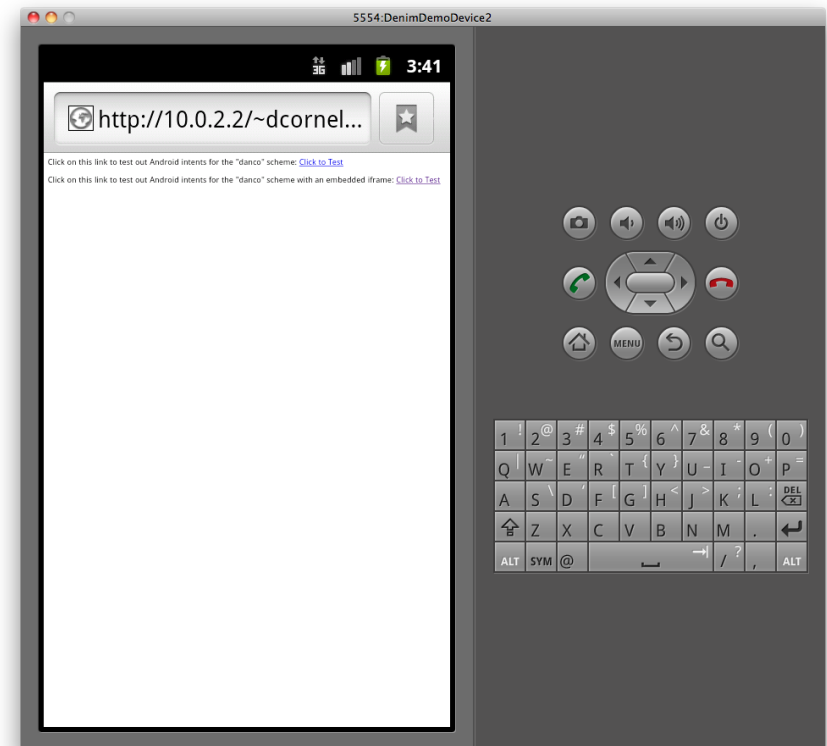
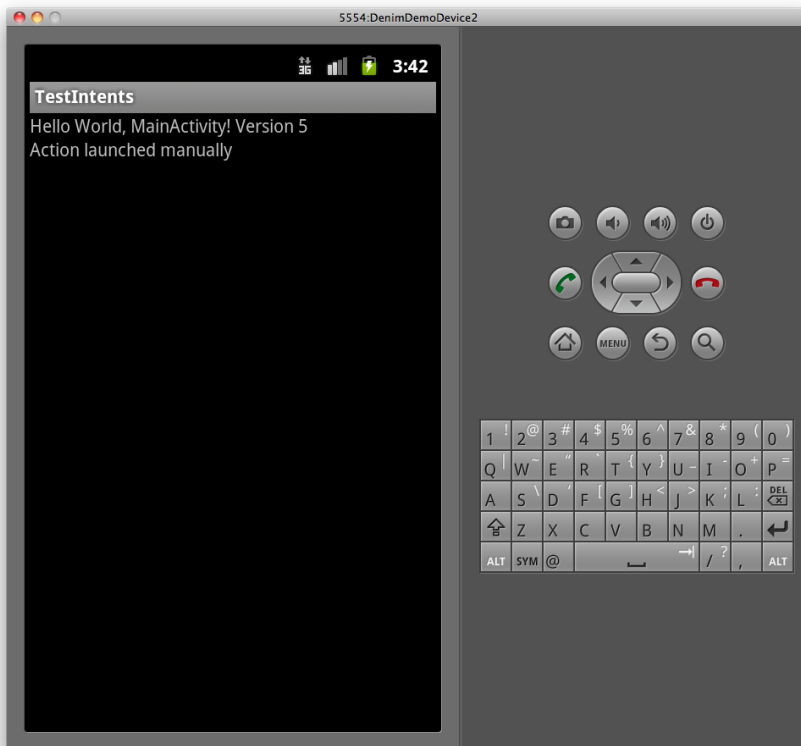
- Intents are facilities for late-binding messaging between applications
 - <http://developer.android.com/guide/topics/intents/intents-filters.html>
- One use is to allow applications to register to receive messages from the Browser when certain types of content are received
 - *Like iOS URL Schemes but an even more comprehensive IPC mechanism*

Intent Filter Example

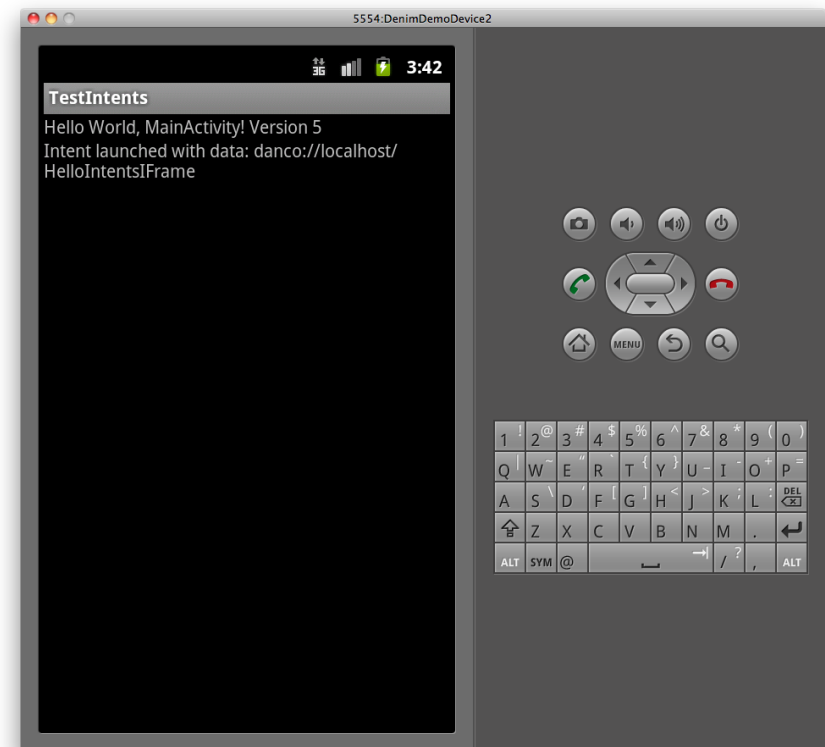
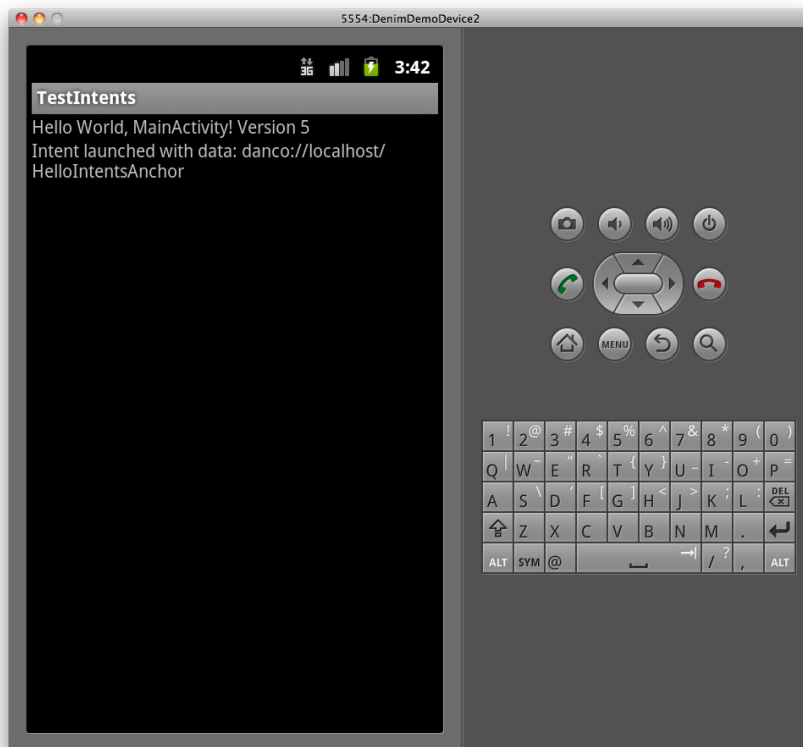
```
<intent-filter>
  <action android:name="android.intent.action.VIEW" />
  <category android:name="android.intent.category.DEFAULT" />
  <category android:name="android.intent.category.BROWSABLE" />
  <data android:scheme="danco" />
</intent-filter>
```

- Action: What to do?
- Data: Scheme is URI “protocol” to handle
- Category BROWSABLE: Allow this Action to be initiated by the browser

Intent Filter Demo – Manual Launch, HTML Page



Intent Filter Demo – Anchor Launch, IFrame Launch



I'm a Security Tester. Why Do I Care?

- URL handlers are remotely-accessible attack surface
- This is a way for you to “reach out and touch” applications installed on a device if you can get a user to navigate to a malicious page
- Send in arbitrary URLs via links or (easier) embedded IFRAMEs
- Example: iOS Skype application used to automatically launch the Skype application and initiate a call when it encountered a skype:// URL
 - *Apple's native Phone handle for <tel://> URLs would confirm before a call was made*

I'm a Developer. Why Do I Care?

- See the previous slide. Bad guys care. So should you. Please.
- Content passed in via these handlers must be treated as untrusted
 - *Positively validate*
 - *Enforce proper logic restrictions*
- All:
 - *Should a malicious web page be able to cause this behavior?*
 - Make phone call, transmit location, take photo, start audio recording, etc
- iOS:
 - *Validate inputs to `handleOpenURL: message`*
- Android:
 - *Validate data brought in from `Action.getIntent()` method*

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”*

Secure Mobile Development Reference

- Platform-specific recommendations
- Key topic areas
- Provide specific, proscriptive guidance to developers building mobile applications

Specific Platforms

- iOS (iPhone, iPad)
 - Android
 - Blackberry (in progress)
 - Windows Phone 7 (in progress)
 - *Windows Mobile 6.5 (?)*
 - Symbian (?)
 - Others (?)
-
- Will be guided by demand, which is focused by new development activity

Topics Areas

- Topic Areas
 - *Overview of Application Development*
 - *Overview of Secure Development*
 - *Defeating Platform Environment Restrictions*
 - *Installing Applications*
 - *Application Permissions Model*
 - *Local Storage*
 - *Encryption APIs*
 - *Network Communications*
 - *Protecting Network Communications*
 - *Native Code Execution*
 - *Application Licensing and Payments*
 - *Browser URL Handling*

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, slides and videos online:

www.smartphonesdumbapps.com

Questions?

Dan Cornell

dan@denimgroup.com

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

www.denimgroup.com

(210) 572-4400