Hidden Gems in APKs

Marc Schönefeld

Agenda

- Motivation
- Foundations of Auditing Android Apps
- Quick walkthrough of APK structure
- 2 case studies (virus scanners, emergency apps)
- Insecure updates
- Tooling
- Summary

Motivation

- Everybody knows that Android application packages (APK) contain code and resources.
- However, some contain information that is designed to be kept private to the application, hence it is encoded or encrypted.

Motivation

- Here, we present a series of failed attempts to keep valuable information from the eye of the interested reverse engineer.
- Often a single fail in APK composition can reveal all useful information contained in the enclosing app.

Motivation

- What can be learned from analyzing the composites of an APK file?
  - Does the vendor catch up with security patches of embedded components?
  - Does the vendor follow secure coding practices?
  - How about fails in using native libraries or encryption?
  - Is the application just a mule for ad placement or mining revenue with privacy data?
- There are many questions that can be answered without or before running the application.

The speaker

- Marc R. Schönefeld
  - Holds a
    - Dr. rer. nat. in Computer Science from the University of Bamberg
    - Master in Business Informatics from the University of Münster
  - Has caused Blue Screens with web fonts
  - Has given security talks & trainings at major conferences
    - Java talks and trainings, plus Reversing Android with Undx,
    - @ PacSec, CanSecWest, Blackhat, HackInTheBox, RSA, J1, Xcon, …
  - Has reported issues in Browsers and operating systems
    - Chrome, IE, Firefox, Safari, Opera, …
  - Loves his Java-related day job with US-based multinational tech corporation
    - however the views expressed here are his own opinion

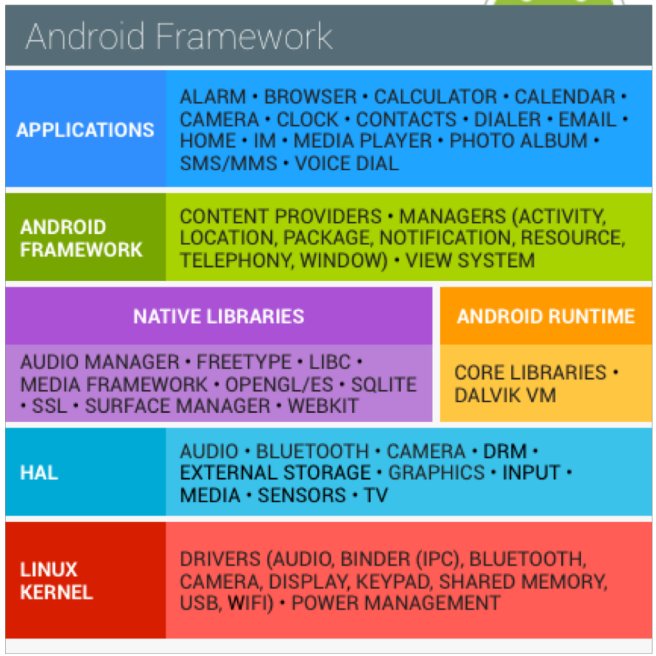
AUDITING ANDROID APPS

Hidden Gems

Android ecosystem

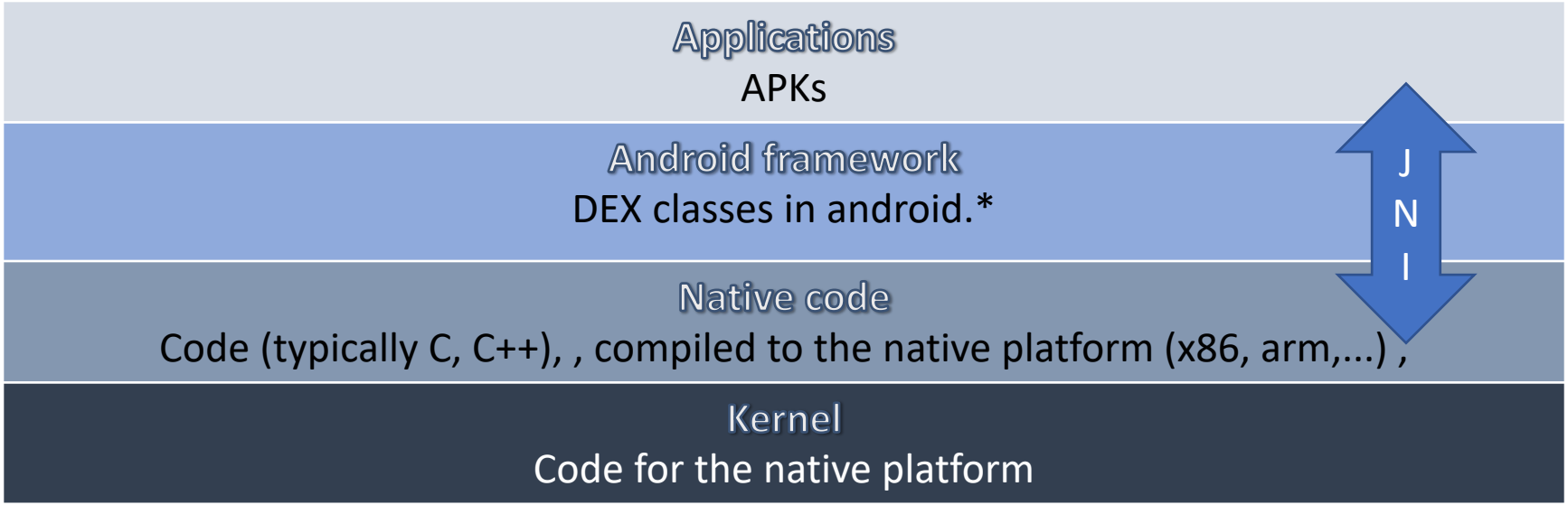
- Android
  - is an open source platform
  - for mobile, embedded and wearable devices
  - is primarily maintained by Google
  - can be customized by device manufacturers to suite needs of their devices
  - uses Play Store as application distribution standard

Android ecosystem



https://source.android.com/security/images/android_software_stack.png]

Android ecosystem



Inspecting Applications for Weaknesses

- Users acquire the majority of mobile applications via download stores. There are two major players on the market, one is the Apple iTunes Store, the other one is the Google PlayStore.
- We chose the Android ecosystem for our investigation, as it provides enough documentation and tools to conduct in-depth technical inspection for a range of criteria, related to IT-security.

But first: What is an APK (quick)?

- An APK is an Android application package
  - You find APKs on Google Play and other app stores
  - Standard packaging format for android apps
  - Technically just a zip, no surprise for those who know jars / crx
  - Contains metadata, for unique identification it has a package name and a version
  - Includes code and configuration, and is signed by author
  - Can be specific for a platform or device type
  - Playstore has own distribution protocol, but third party stores allow direct http(s)-Download

Terminology

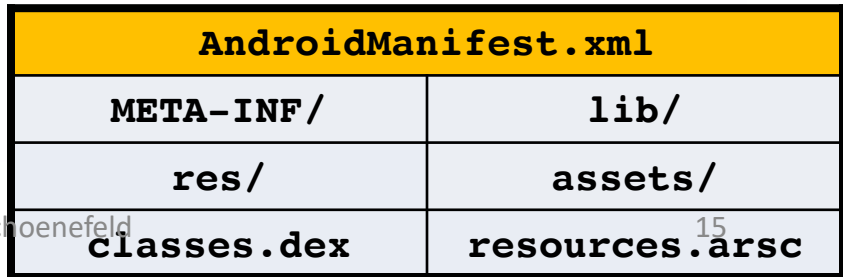
- Gold digging in APKs
- What can become valuable as gold for a security researcher?
  - Credentials
  - App Code that is vulnerable or can be repurposed, debug information
  - Vulnerable dependencies and 3rd party components
  - Information helpful for social engineering
- Inspection for these can be done by combining standard reverse engineering tools, such as Qark, Apktool, Exodus (for Trackers), ScanAPK and radare.

But first: What is an APK (quick)?

AndroidManifest.xml	
META-INF/	lib/
res/	assets/
classes.dex	resources.arsc

APK standard file–AndroidManifest.xml

- The AndroidManifest.xml (binary XML) describes
  - Top-level information
    - name, version, permissions and interfaces of the applications.
  - Internal and exported interfaces
  - Services, Activities, Receivers, Intents
    - Name, Parameters (exported /internal)
  - Secret numbers
  - Network configurations
  - …



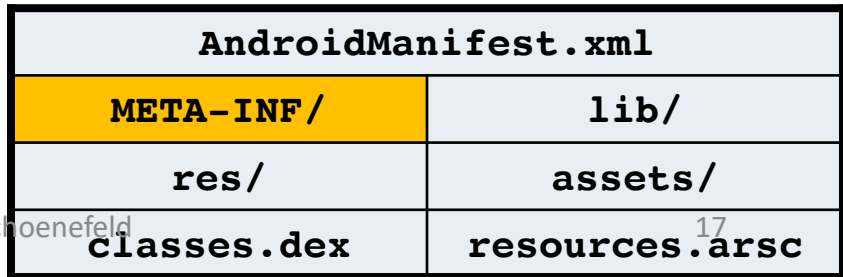
APK standard file–AndroidManifest.xml

- What gold to find in AndroidManifest.xml?
  - Backup enabled?
  - Is App Debuggable?
  - Are there exported components which should be kept internal?
  - Credentials
    - Passwords, private keys in cleartext or obfuscated strings
  - More Strings/URLs to expose secrets
    - Information about infrastructure (hostnames)

APK standard directory – META-INF

First area of APK which checked by the runtime

- Adds integrity to the Zip container
- Android has history of APK parsing problems
  - MasterKey bug verified first occurrence and installed second (CVE-2013-4787)

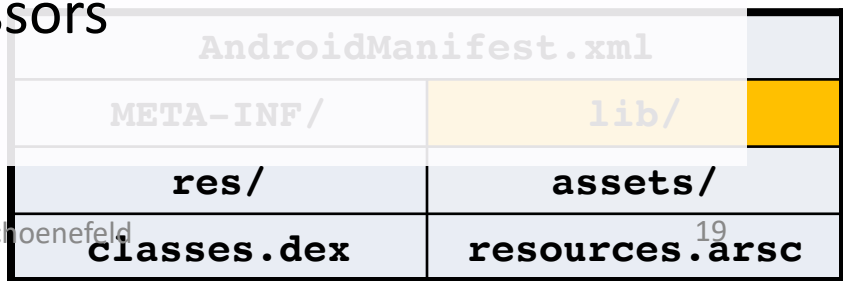


APK standard directory – META-INF

- What gold to expect in “META-INF” ?
- Certificate (*.RSA) may reveal information about the author
  - Self-signed
  - Matches app description?
  - Same as signature in other packages of the vendor?
  - Proper Key length, expiration?
- The Manifest itself
  - A full list of files of the package
  - Checksums allow to detect changes without have to compare files

APK standard directory – lib

- The lib directory contains native libraries, which is compiled for the processor of the device.
- There are one or multiple subdirectories:
  - armeabi: for all ARM based processors
  - armeabi-v7a: for all ARMv7 and above based processors
  - x86: for x86 processors
  - x86_64: for x86 (64-bit) processors
  - mips: for MIPS processors



APK standard directory – lib

- What gold to expect in “lib” ?
- Java Native Interface
  - For a variety of C packages there are JNI bindings for accessibility from Java (NDK)
  - Some applications also move the important code parts away from managed code to native code to deal with easy bytecode decompilation
- Framework support code (.NET, XCrossWalk)

APK standard directory – lib

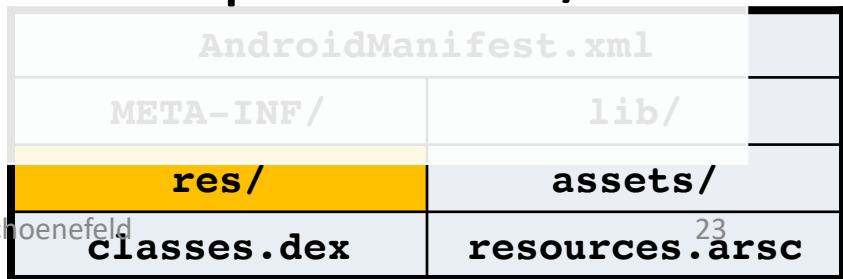
- Native libraries tend to outdate, often found are:
  - OpenSSL,
  - OpenCV,
  - SQLite,
  - LibPng, libjpeg, libxml, expat, ….
- Builds
  - Insecure compile and link
  - Leftover strings in binaries
- Some vendors decided to hide more native code elsewhere (such as Facebook in xz-Archives located in res, Mozilla in assets directory)

Where to find native code in APKs

- Native code resides in .so (shared object) and executable files, you can find those
  - In the lib directory
  - also in the assets directory, or in res/raw
  - Or multiple of those in an archive (zip/7z/xz/...)
- Often each supported architecture has it‘s own native library subdirectory
  - armeabi, armeabi-v7a, arm64-v8a, mips, mips64, x86, x86_64,
- Preinstalled apps also have their libs in /usr/lib

APK standard directory – res

- The res directory contains animations, color definitions, drawables, layouts, menus, and xml
- Resources are numbered (R.java)
- Also contains the infamous dump area res/raw

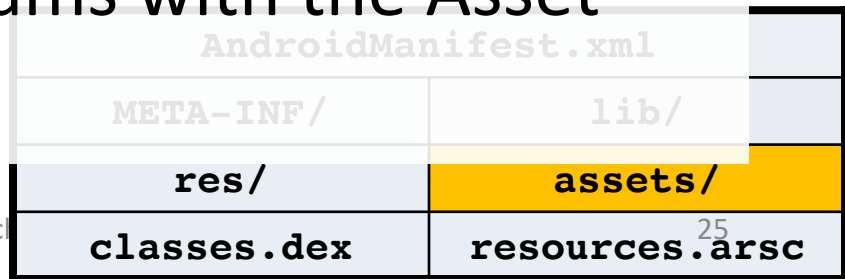


APK Standard directory – res

- What gold to expect in “res” ?
  - A lot of files in binary xml
    - (apktool/aapt can help here)
  - res/raw
    - Here you can find all kinds of interesting things
    - Some apps have over 100 files here
  - Great place to hide files, ymmv

APK standard directory – assets

- While resources are numbered, assets can be found with a file URI
- Original file name is preserved
- Typical file types are HTML, JS , CSS, where file name maybe important
- To be accessed as byte streams with the Asset Manager

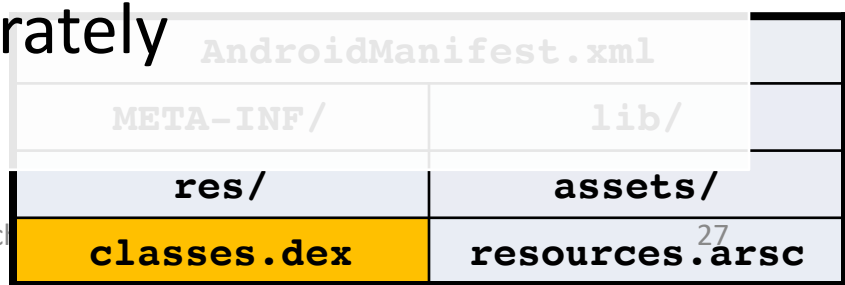


APK standard directory – assets

- What gold to expect in “assets” ?
- Can find out about reachable URLs from WebView.
- Vulnerable JS components?
  - Outdated Jquery?
  - Outdated framework code like Cordova?

APK standard file – classes.dex

- Typically from Java source code (or Kotlin,…)
- Bytecode in dex format (Dalvik Executable)
- Can be multidex, when there are over 65536 methods
  - multiple files, classes2.dex, classes3.dex, etc.
  - apktool stores theses separately (smali_Classes2,…)



APK standard file – classes.dex

- What gold to expect in “classes.dex” ?
  - Outdated vulnerable Java components?
    - Like OkHTTP, Apache HTTP
    - Bouncy/Spongycastle
  - Tracker & Adware code (signatures from Exodus project)
  - Credentials in code such as
    - Passwords, private keys in cleartext or obfuscated strings

APK standard file – classes.dex

- What gold to expect in “classes.dex” (contd.)?
  - Programming
    - Insecure crypto
    - Insecure updates (data or code)
    - Dangerous API , Suppressed Lint Warnings (@SuppressLint)

APK standard file – resources.arsc

- resources.arsc is a compiled resource file
- Typical resources types:
  - Strings
  - Dimensions
- Binary format,
  - can be printed with aapt,
  - apktool stores it readable in strings.xml, arrays.xml, etc.

APK standard file – resources.arsc

- What gold to expect in “resources.arsc” ?
  - Strings may contain
    - credentials (cleartext or obfuscated)
    - Hostnames
    - Service identifiers
    - IP-Addresses
  - IDs can lead you from the resource to the code accessing it

Code areas in APKs

- Identifying vulnerable code areas
  - Code type
    - Java Code -> classses.dex
    - **Native code (standalone, JNI**) -> lib, res
    - JavaScript executed in WebView -> assets
  - Frequent Weakness types
    - **Missing binary protections**
    - **Outdated 3rd party libraries**
    - **Crypto usage**
      - **Hardcoded keys, Trustmanager, Hostnameverifier, …**

Top 10 2013-A9-Using Components with Known Vulnerabilities

← A8-Cross-Site Request Forgery (CSRF)

2013 Table of Contents

2013 Top 10 List

A10-Unvalidated Redirects and Forwards →

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
Application Specific	Exploitability AVERAGE	Prevalence WIDESPREAD	Detectability DIFFICULT	Impact MODERATE	Application / Business Specific
Some vulnerable components (e.g., framework libraries) can be identified and exploited with automated tools, expanding the threat agent pool beyond targeted attackers to include chaotic actors.	Attacker identifies a weak component through scanning or manual analysis. He customizes the exploit as needed and executes the attack. It gets more difficult if the used component is deep in the application.	Virtually every application has these issues because most development teams don't focus on ensuring their components/libraries are up to date. In many cases, the developers don't even know all the components they are using, never mind their versions. Component dependencies make things even worse.		The full range of weaknesses is possible, including injection, broken access control, XSS, etc. The impact could range from minimal to complete host takeover and data compromise.	Consider what each vulnerability might mean for the business controlled by the affected application. It could be trivial or it could mean complete compromise.

Am I Vulnerable To 'Using Components with Known Vulnerabilities'?

In theory, it ought to be easy to figure out if you are currently using any vulnerable components or libraries. Unfortunately, vulnerability reports for commercial or open source software do not always specify exactly which versions of a component are vulnerable in a standard, searchable way. Further, not all libraries use an understandable version numbering system. Worst of all, not all vulnerabilities are reported to a central clearinghouse that is easy to search, although sites like CVE and NVD are becoming easier to search.

Determining if you are vulnerable requires searching these databases, as well as keeping abreast of project mailing lists and announcements for anything that might be a vulnerability. If one of your components does have a vulnerability, you should carefully evaluate whether you are actually vulnerable by checking to see if your code uses the part of the component with the vulnerability and whether the flaw could result in an impact you care about.

How Do I Prevent 'Using Components with Known Vulnerabilities'?

One option is not to use components that you didn't write. But that's not very realistic.

Most component projects do not create vulnerability patches for old versions. Instead, most simply fix the problem in the next version. So upgrading to these new versions is critical. Software projects should have a process in place to:

1. Identify all components and the versions you are using, including all dependencies. (e.g., the versions plugin).
2. Monitor the security of these components in public databases, project mailing lists, and security mailing lists, and keep them up to date.
3. Establish security policies governing component use, such as requiring certain software development practices, passing security tests, and acceptable licenses.
4. Where appropriate, consider adding security wrappers around components to disable unused functionality and/ or secure weak or vulnerable aspects of the component.

Example Attack Scenarios

Component vulnerabilities can cause almost any type of risk imaginable, ranging from the trivial to sophisticated malware designed to target a specific organization. Components almost always run with the full privilege of the application, so flaws in any component can be serious, The following two vulnerable components were downloaded 22m times in 2011.

- Apache CXF Authentication Bypass – By failing to provide an identity token, attackers could invoke any web service with full permission. (Apache CXF is a services framework, not to be confused with the Apache Application Server.)
- Spring Remote Code Execution – Abuse of the Expression Language implementation in Spring allowed attackers to execute arbitrary code, effectively taking over the server.

Every application using either of these vulnerable libraries is vulnerable to attack as both of these components are directly accessible by application users. Other vulnerable libraries, used deeper in an application, may be harder to exploit.

References

OWASP

- OWASP Dependency Check (for Java libraries)
- OWASP SafeNuGet (for .NET libraries thru NuGet)

External

- The Unfortunate Reality of Insecure Libraries
- Open Source Software Security
- Addressing Security Concerns in Open Source Components
- MITRE Common Vulnerabilities and Exposures
- Example Mass Assignment Vulnerability that was fixed in ActiveRecord, a Ruby on Rails GEM

https://www.owasp.org/index.php/Top_10_2013-A9-Using_Components_with_Known_Vulnerabilities

OWASP and vulnerable components

- *Virtually every application has these issues because most development teams don't focus on ensuring their components/libraries are up to date.*
- *In many cases, the developers don't even know all the components they are using, never mind their versions.*
- *Component dependencies make things even worse.*

https://www.owasp.org/index.php/Top_10_2013-A9-Using_Components_with_Known_Vulnerabilities

Is my app vulnerable?

- You are likely vulnerable:
  - If you **do not know the versions** of all components you use […] directly [..] as well as nested dependencies.
  - If any of your software **out of date**? This includes […] all components, runtime environments and libraries.
  - If you **do not know if they are vulnerable**. […]
  - If you **do not fix nor upgrade** […] in a timely fashion.

https://github.com/OWASP/Top10/blob/master/2017/OWASP%20Top%2010%202017%20RC2%20Final.pdf

MINI CASE STORY: SECURITY TOOLS FROM 2018

Samples

- Antivirus apps from the Android Playstore
  - Expected to be spearhead of security awareness
  - Use managed and native code
  - Use cryptography in secure communications
  - Have engine updates from their backend services
- Underwent tool-assisted check for insecurity patterns
- Sample size is five apps
- the first apps listed a-z on av-test.org were chosen

Research Objects

NAME	GOOGLE PLAY ID	LAST UPDATE
Mobile Security	**com.ali.money.shield**	Apr 27
V3Mobile Security	**com.ahnlab.v3mobilesecurity.soda**	Nov 19
AVL	**com.antiy.avl**	Oct 31
Avast	**com.avast.android.mobilesecurity**	Nov 23
Avira Antivirus	**com.avira.android**	Nov 23

Note: To focus on the bugs instead of products, in the following slides the sequence was randomized / anonymized

Typical Weaknesses

Weakness	Description
CWE-89	SQL-Injection (CIA)
CWE-200	Information exposure (C)
CWE-250	Execution with unnecessary Privileges (CIA)
CWE-256	Cleartext passwords (C)
CWE-295	Improper certificate validation (CI)
CWE-311	Missing encryption of sensitive data (C)
CWE-693	Protection Mechanism Failure (I)
CWE-937	Components with known vulnerabilities (CIA)

Results App 1

- CWE-89: SQL queries not always built with a CompiledStatement, instead query string composed via String concatenation
- CWE-250: Dangerous Permissions, can record audio, send/receive SMS, read/modify call log, read/write contacts, process outgoing calls
- CWE-256: Cleartext API Key to own cloud service in Resources
- CWE-693: Missing stack canaries
- CWE-937: Bundles outdated expat-2.1.0 and LibreSSL 2.5.5
- Product has over 10 Trackers: Adjust, Google CrashLytics, Facebook Ads, FB Analytics, FB Login, FB Places, FB Share, G Ads, G Doubleclick, G Firebase Analytics, MixPanel

Results App 2

- CWE-250: Dangerous Permissions, can send/receive SMS, process outgoing calls, record audio, read/modify call log, access phone, read/write contacts and others
- CWE-311: Embedded update APK uses http URLs to backend (MITM)
- CWE-693:
  - Missing stack canaries and pic settings for so files
  - BusyBox binary has no protections
- CWE-937:
  - Bundles outdated OpenSSL 1.0.2j from 2016
  - BusyBox 1.17.2 from August 2010 (CVE-2018-1000517) and IPTables of unknown version
- Trackers: AppsFlyer, CrashLytics, FB Ads, FB Analytics, FB Share, G Ads, G DoubleClick, G Analytics, G Firebase Analytics, Inmobi, Twitter MoPub

App2: Bundled insecure executables

- This app ships with standalone binaries of Busybox and IPTables
  - Upon request vendor was not able to describe what these are used for
  - Side note: Both GPL-licensed binaries do not seem to have source attached, but this might open an entirely different can of worms
  - These binaries also had insecure compile/link settings

App2: Bundled insecure executables

```
/res/raw/busybox_g1
===================
busybox=BusyBox v1.17.2 (2010-09-24 12:09:05 BRT)
SHA256=3393e852a09336245d5a03122699afab2f7e9adb2f36c5c88161644
c4cd1c4e1
Size=117256
Header (r2) bin=
{"arch":"arm","binsz":116575,"bintype":"elf","bits":32,"canary
":false,"class":"ELF32","compiled":"","crypto":false,"dbg_file
":"","endian":"little","havecode":true,"guid":"","intrp":"","l
ang":"c","linenum":false,"lsyms":false,"machine":"ARM","maxops
z":4,"minopsz":4,"nx":false,"os":"linux","pcalign":4,"pic":fal
se,"relocs":false,"rpath":"NONE","static":true,"stripped":true
,"subsys":"linux","va":true,"checksums":{}}
Build exception= canary, nx, pic
```

Results App 3

- CWE-250: Needs permissions to call with the phone, read phone stats
- CWE-321:
  - Hardcoded PrivateKey
  - Hardcoded SecretKey
- CWE-937:
  - Bundles outdated OpenSSL 1.0.2j from 2016
- No known trackers

Old OpenSSL in App 3

```
======================
/lib/armeabi/libavlm.so
======================
GCC: (GNU) 4.9.x 20150123 (prerelease), Unknown
openssl=OpenSSL 1.0.2j  26 Sep 2016
Suspicious calls=['sprintf', 'strcat', 'strcpy', 'strcmp']
SHA256=333410b9c468fe1638711726b2f38adff5198f3df0d14d5fcb10fff79316e904
Size=1824240
Header (r2)
bin={"arch":"arm","binsz":1823280,"bintype":"elf","bits":32,"canary":true,"cla
ss":"ELF32","compiled":"","crypto":false,"dbg_file":"","endian":"little","have
code":true,"guid":"","intrp":"","lang":"cxx","linenum":false,"lsyms":false,"ma
chine":"ARM","maxopsz":4,"minopsz":4,"nx":true,"os":"linux","pcalign":4,"pic":
true,"relocs":false,"relro":"full","rpath":"NONE","static":true,"stripped":tru
e,"subsys":"linux","va":true,"checksums":{}}
```

Hardcoded secret Key in App 3

```
Key secretKeySpec = new SecretKeySpec(b(new byte[]{(byte)
118, …., (byte) 10}), "AES");
try {
    Cipher instance = Cipher.getInstance(new
String(b(bArr2), "UTF-8"));
    instance.init(2, secretKeySpec);
    return new String(instance.doFinal(bArr), "UTF-8");
}
catch (UnsupportedEncodingException e) {….}
```

- method b is a 0x80 xor over the byte array

Search for MIIE … and the key is yours

Hiding Private Keys, but leaving the MIIE in:

```
public final class l {
  String a =   "MIIEv….".concat("….jYt").concat("….w=");
    public static RSAPrivateKey a() {
        try {
            return (RSAPrivateKey) KeyFactory.getInstance("RSA").
                                generatePrivate(new
PKCS8EncodedKeySpec(Base64.decode(a, 0)));
        } catch (…}
     return null;
  }
```

jshell & OpenSSL to infer key type

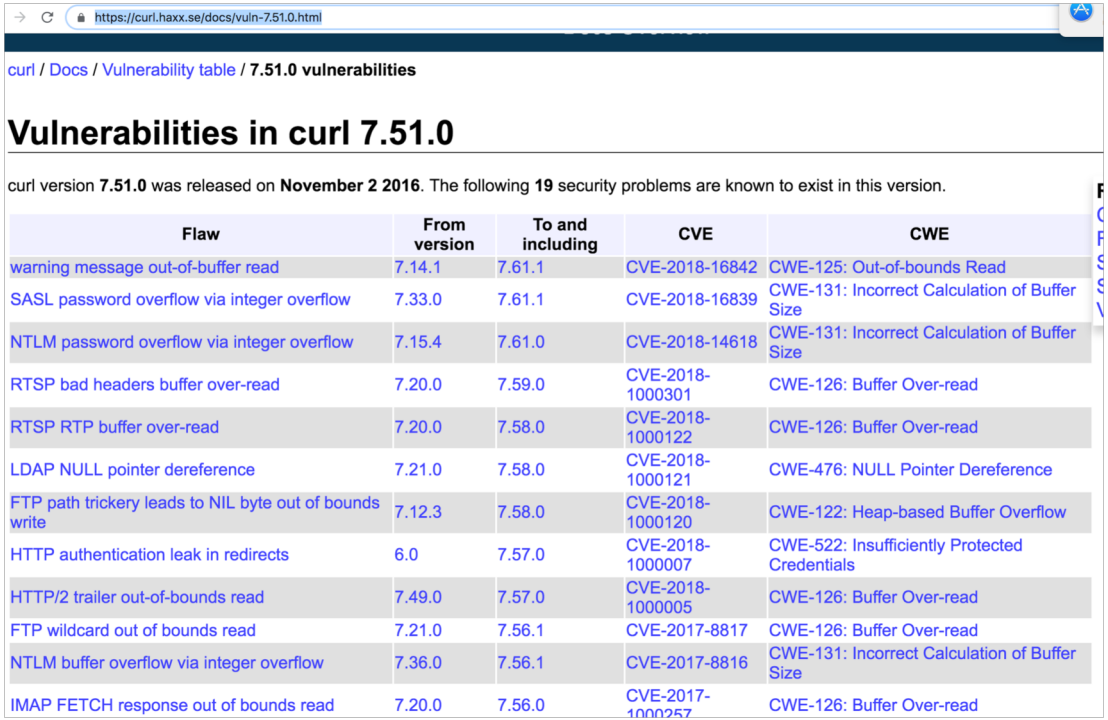
```
jshell>  String a =   "MIIEv….".concat("….jYt").concat("….w=");
a ==> "MIIE ..nnw="
jshell> var b = new FileOutputStream("somekey.txt")
b ==> java.io.FileOutputStream@4cf777e8
jshell> b.write(a.getBytes())
jshell> b.close()

base64 -D <somekey.txt | openssl rsa -inform DER
  -check -text  | grep -i RSA
    writing RSA key
    RSA key ok
    -----BEGIN RSA PRIVATE KEY-----
    -----END RSA PRIVATE KEY-----
```

Results App 4

- CWE-250: Dangerous Permissions: Full access to SMS, reading phonestats call log and contacts
- CWE-311: Accesses multiple webservices in backend with http (MITM)
- CWE-321: Hardcoded secret key
- CWE-693: Insecure native build settings (canaries)
- CWE-937: Native libs are outdated
  - OpenAL 1.12.584 from 2013
  - OpenSSL 1.0.1u from Sep 2016
  - zlib 1.2.5 from April 2010
  - Curl 7.51.0 from November 2016
- Tracker signatures: Google CrashLytics,Facebook Login,FB Places,FB Share,G Ads,G DoubleClick,G Analytics,G Firebase Analytics

App 4



Flaw	From version	To and including	CVE	CWE
warning message out-of-buffer read	7.14.1	7.61.1	CVE-2018-16842	CWE-125: Out-of-bounds Read
SASL password overflow via integer overflow	7.33.0	7.61.1	CVE-2018-16839	CWE-131: Incorrect Calculation of Buffer Size
NTLM password overflow via integer overflow	7.15.4	7.61.0	CVE-2018-14618	CWE-131: Incorrect Calculation of Buffer Size
RTSP bad headers buffer over-read	7.20.0	7.59.0	CVE-2018-1000301	CWE-126: Buffer Over-read
RTSP RTP buffer over-read	7.20.0	7.58.0	CVE-2018-1000122	CWE-126: Buffer Over-read
LDAP NULL pointer dereference	7.21.0	7.58.0	CVE-2018-1000121	CWE-476: NULL Pointer Dereference
FTP path trickery leads to NIL byte out of bounds write	7.12.3	7.58.0	CVE-2018-1000120	CWE-122: Heap-based Buffer Overflow
HTTP authentication leak in redirects	6.0	7.57.0	CVE-2018-1000007	CWE-522: Insufficiently Protected Credentials
HTTP/2 trailer out-of-bounds read	7.49.0	7.57.0	CVE-2018-1000005	CWE-126: Buffer Over-read
FTP wildcard out of bounds read	7.21.0	7.56.1	CVE-2017-8817	CWE-126: Buffer Over-read
NTLM buffer overflow via integer overflow	7.36.0	7.56.1	CVE-2017-8816	CWE-131: Incorrect Calculation of Buffer Size
IMAP FETCH response out of bounds read	7.20.0	7.56.0	CVE-2017-1000257	CWE-126: Buffer Over-read

Hardcoded secret keys (CWE-321)

```
public class Config { // removed all other keys
    public  static final String ENC_SUPER_KEY = "aW*=";
}
```

Later:

```
key = new SecretKeySpec(Ig.decode(Config.ENC_SUPER_KEY).getBytes(),
"AES");
Cipher instance = Cipher.getInstance("AES/ECB/PKCS5PADDING");
instance.init(2, key);
return new String(instance.doFinal(ig.decode(str)));
```

The ENC_SUPER_KEY also starts with 8 bytes of the company name, which could shorten brute force attempts by a significant time

Results App 5

- CWE-250: Dangerous Permissions
  - full access to SMS, contacts, call log and calendar
- CWE-295: Weak HostNameVerifier used
- CWE-311: Insecure JavaScript downloads (MITM)
- CWE-321
  - Hmac with hardcoded key
  - Hardcoded Secret Key
- CWE-693: Insecure build settings (canaries,nx,relro,pic)
- No tracker signatures found

- Plus: Contains URLs for further APKs, possibly bypassing Play Store
  - CWE-937: Expat 2.1.0, LibPng 1.6.22, libxml 2.9.4

App 5: CWE-311
Insecure Javascript download

App 5: Weak hostname verifier

```
private static class a implements HostnameVerifier {

    public boolean verify(String str, SSLSession s) {
            return true;
    }

}
```

App 5: HMAC with known key

```
 public static String calcHmac(byte[] bArr) throws
Exception {
       byte[] bArr2 = new byte[]{69,  …. , 38, 93};
       Mac inst = Mac.getInstance("HmacSHA1");
       inst.init(new SecretKeySpec(h.a(bArr2),
inst.getAlgorithm()));
       return inst.doFinal(bArr);
}
```

App 5: empty IV/hardcoded secret key

```
byte[] a(byte[] bArr) throws Exception {
  var iv = new IvParameterSpec(
    new byte[]{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0});
  var secretKeySpec = new SecretKeySpec(
      new byte[]{38, 40,…, 61, 61, 61}, "AES");
  var i= Cipher.getInstance("AES/CBC/PKCS5Padding");
  i.init(1, secretKeySpec, iv);
  return i.doFinal(bArr);
}
```

Fwiw, Framework name appears in the key string

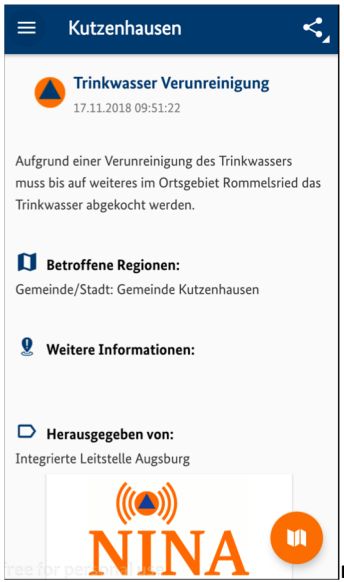
Results: Security Tools

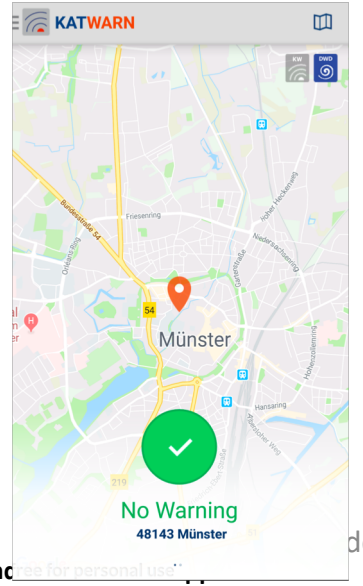
Application	CWE-89	CWE-250	CWE-256	CWE-295	CWE-311	CWE-321	CWE-693	CWE-937	Trackers
App 1	(+)	(+)	(+)				(+)	(+)	11
App 2		(+)			(+)		(+)	(+)	11
App 3		(+)				(+)		(+)	0
App 4		(+)			(+)	(+)	(+)	(+)	8
App 5		(+)		(+)	(+)	(+)	(+)	(+)	0

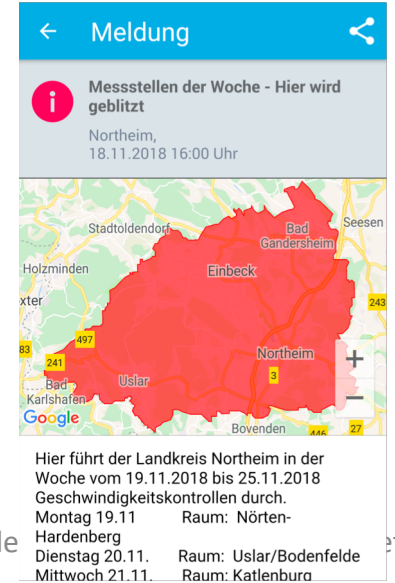
MINI CASE STUDY 2: EMERGENCY ALERT APPS

NINA

- 5 Emergency Alert Apps from DE and CH
- checked for security issues (according to CWE)







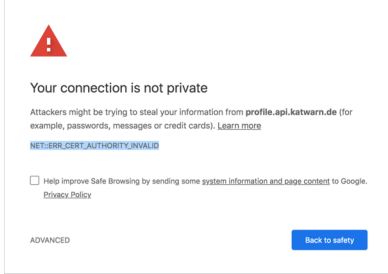
Research Objects

NAME	GOOGLE PLAY ID	LAST UPDATE
NINA	de.materna.bbk.mobile.app	Sep 18, 2018
KATWARN	de.combirisk.katwarn	Nov 22, 2017
BIWAPP	de.mplg.biwapp	Aug 17, 2018
Warnwetter	de.dwd.warnapp	July 19, 2018
AlertSwiss	ch.admin.babs.alertswiss	Nov 13, 2018

Results App 1

- CWE-200: Exports receiver, protected with custom permission, could have been obtained earlier by malicious app (Murphy, 2015)
- CWE-250: READ_EXTERNAL_STORAGE and WRITE_EXTERNAL_STORAGE permissions are dangerous (injection)
- CWE-311: References own homepage via HTTP. This could be misused in a man-in-the-middle attack (such as malicious redirect)
- CWE-937: Bundles outdated amazonaws-adapted copy of Apache HTTPClient 4.2.3 → CVE-2013-4366 (cvss2=7.5, CWE-20)
- No tracker signatures found
- Developer contact address invalid as of Nov 2018

Results App 2



- CWE-89: ExecSQL not always built with a CompiledStatement, often query string composed via String concatenation, which may give way to SQL injection
- CWE-295: Connects to multiple API endpoints which have self-signed certs (NET::ERR_CERT_AUTHORITY_INVALID)
- CWE-311: The PageIndicatorGridView uses a HTTP URL, which can allow MITM-scenarios
- Needs Play Store Services
- Signatures of Google Ads, Doubleclick, and Analytics found

Results App 3

- CWE-256: App shares API Key & password to backend service
  - Potential misuse of these credentials may impact infrastructure
- Vendor informed in early 2018
  - claims this works as designed, and stays unchanged in current version
  - but will go away with upcoming major version
- Requires Google Play Services, has Google Analytics Tracker

Results App 4

- CWE-937:
  - Bundles LibXml 2.9.4 from 2016 in embedded native library
  - Bundles SQLite 3.17.0
  - Bundes OpenSSL and LibXslt of unknown version
- Needs Google Play Services
- Trackers: Google Firebase Analytics, Acra, Matomo (Piwik)

Results App 5

- CWE-250: Implementation of parts of the functionality in native code (JNI libraries). Usage of managed code (Java/Kotlin) instead would be inherently safer
- CWE-937: Outdated components
  - Native SQLite 3.8.2 implementation comes from 2013, and is affected by a wide range of CVE-listed vulnerabilities
  - Bundles obfuscated copy of Apache HTTP of unknown version
- Tracks with Google Analytics, Google Firebase Analytics and Acra

Results Overview

Application	CWE-89	CWE-200	CWE-250	CWE-256	CWE-295	CWE-311	CWE-937	Trackers
App 1		(+)	(+’)			(+)	(+)	0
App 2	(+)				(+)	(+)		3
App 3				(+)				1
App 4							(+)	3
App 5			(+)				(+)	3

INSECURE ENGINE UPDATE

Insecure engine update

- It is a frequent requirement to update code or data of an app:
  - „We strongly discourage loading code from outside of your application APK.
  - Doing so significantly increases the likelihood of application compromise due to code injection or code tampering.
  - It also adds complexity around version management and application testing.
  - It can also make it impossible to verify the behavior of an application, so it may be prohibited in some environments.“

https://developer.android.com/training/articles/security-tips.html

Insecure engine update

- However, some apps don‘t care
- The following example shows the insecure update of a virus scanner with 1.000.000+ downloads.
  - Which bypasses the Play Store
  - Over an unprotected connection
  - Missing a proof-of-origin (however with an MD5 ☺)
  - Containing native shared libraries

Engine update in plaintext with executable content

```
public class DataUpdater implements DataDownloadListener {
   String SERVER_INFOFILE_URL =
     "http://download.c*******y.com/data_info_cfg.xml";
     [...]

   void updateNativeLibs(Context v1, File v2) {
      Iterator v4=this.getUpdateList().iterator();
      while(v4.hasNext()) {
        DataLibInfo v5 = (DataLibInfo)v4.next();
             [...]
```

Engine update in plaintext with executable content

```
wget -O - http://download.c*******y.com/data_info_cfg.xml
--2018-08-21 09:23:06--  http://download.c*******y.com/data_info_cfg.xml
Resolving download.c********y.com (download.c*******y.com)... 5*.**.248.95, 5*.**.14.164
[…]

<data mainVersion="15000416">
  <lib>
    <id>avbases</id>
    <version>20150004161900</version>
    <country>global</country>
    <size>1003717</size>
    <type>zip</type>
    <compressed>true</compressed>
    <silence>false</silence>
    <url>http://download.c********y.com/15000416/avbases.zip</url>
    <md5>28c2b682b2136a0f55d3fbbc59ca1a90</md5>
  </lib>
</data>
```

Engine update unsigned with executable content

```
wget -O - http://download.c*******y.com/data_info_cfg.xml
--2018-08-21 09:23:06--  http://download.c*******y.com/data_info_cfg.xml
Resolving download.c********y.com (download.c*******y.com)... 5*.**.248.95, 5*.**.14.164
[…]

<data mainVersion="15000416">
  <lib>
    <id>avbases</id>
    <version>20150004161900</version>
    <country>global</country>
    <size>1003717</size>
    <type>zip</type>
    <compressed>true</compressed>
    <silence>false</silence>
    <url>http://download.c********y.com/15000416/avbases.zip</url>
    <md5>28c2b682b2136a0f55d3fbbc59ca1a90</md5>
  </lib>
</data>
```

Engine update in plaintext with executable content

```
http://download.c*********y.com/15000416/avbases.zip
--2018-08-21 07:58:54--  http://download.c*********y.com/15000416/avbases.zip
Resolving download.c*********y.com (download.c*********y.com).. 5*.**.248.95

[…]

Saving to: 'avbases.zip'

avbases.zip    100%[=====================================>] 982.23K   357KB/s
```

Update package contains native libraries

```
unzip -t avbases.zip
Archive:  avbases.zip
    testing: avbases/                   OK
    testing: avbases/android.kdc        OK
    testing: avbases/index              OK
    testing: avbases/kavheur.kdl        OK
    testing: avbases/kms90.avc          OK
    testing: avbases/libkavheur.kdl.so   OK
    testing: avbases/libkavsdk.so       OK
    testing: avbases/list.ksl           OK
    testing: avbases/mmh001.kdc         OK
    testing: avbases/mmh002.kdc         OK
    testing: avbases/mmheur.mft         OK
    testing: avbases/mmheur01.kdc       OK
    testing: avbases/mmhlnk01.kdc       OK
    testing: avbases/version.txt        OK
No errors detected in compressed data of avbases.zip.
```

Update package contains native libraries

```
unzip -t avbases.zip
Archive:  avbases.zip
    testing: avbases/                   OK
    testing: avbases/android.kdc        OK
    testing: avbases/index              OK
    testing: avbases/kavheur.kdl        OK
    testing: avbases/kms90.avc          OK
    testing: avbases/libkavheur.kdl.so   OK
    testing: avbases/libkavsdk.so       OK
    testing: avbases/list.ksl           OK
    testing: avbases/mmh001.kdc         OK
    testing: avbases/mmh002.kdc         OK
    testing: avbases/mmheur.mft         OK
    testing: avbases/mmheur01.kdc       OK
    testing: avbases/mmhlnk01.kdc       OK
    testing: avbases/version.txt        OK
No errors detected in compressed data of avbases.zip.
```

Secure binaries, so what?

```
==========================
/avbases/libkavheur.kdl.so
==========================
GCC: (GNU) 4.4.3, Unknown
Suspicious calls=['strcmp', 'sprintf', 'strcpy']
SHA256=c7687adc4ec77223afcbd81f12402dc2453349b1c22d30df1fcadd3b01708e74
Size=796804
Header (r2) bin=
{"arch":"arm","binsz":795922,"bintype":"elf","bits":32,"canary":false,"class":
"ELF32","compiled":"","crypto":false,"dbg_file":"","endian":"little","havecode
":true,"guid":"","intrp":"","lang":"cxx","linenum":false,"lsyms":false,"machin
e":"ARM","maxopsz":4,"minopsz":4,"nx":false,"os":"linux","pcalign":4,"pic":tru
e,"relocs":false,"relro":"no","rpath":"NONE","static":true,"stripped":true,"su
bsys":"linux","va":true,"checksums":{}}
Build exception= canary,nx,relro
```

Secure binaries, so what?

```
==========================
/avbases/libkavheur.kdl.so
==========================
GCC: (GNU) 4.4.3, Unknown
Suspicious calls=['strcmp', 'sprintf', 'strcpy']
SHA256=c7687adc4ec77223afcbd81f12402dc2453349b1c22d30df1fcadd3b01708e74
Size=796804
Header (r2) bin=
{"arch":"arm","binsz":795922,"bintype":"elf","bits":32,"canary":false,"class":
"ELF32","compiled":"","crypto":false,"dbg_file":"","endian":"little","havecode
":true,"guid":"","intrp":"","lang":"cxx","linenum":false,"lsyms":false,"machin
e":"ARM","maxopsz":4,"minopsz":4,"nx":false,"os":"linux","pcalign":4,"pic":tru
e,"relocs":false,"relro":"no","rpath":"NONE","static":true,"stripped":true,"su
bsys":"linux","va":true,"checksums":{}}
Build exception= canary,nx,relro
```

Secure binaries, so what?

```
====================
/avbases/libkavsdk.so
====================
GCC: (GNU) 4.4.3, Unknown
Suspicious calls=['strcpy', 'strcat', 'strcmp', 'sprintf']
SHA256=9e97f952729014575da9eb783c6d772f7d0ad10b27b8cb523d2d103f822db1ec
Size=186312
Header (r2) bin=
{"arch":"arm","binsz":185430,"bintype":"elf","bits":32,"canary":false,"class":
"ELF32","compiled":"","crypto":false,"dbg_file":"","endian":"little","havecode
":true,"guid":"","intrp":"","lang":"cxx","linenum":false,"lsyms":false,"machin
e":"ARM","maxopsz":4,"minopsz":4,"nx":false,"os":"linux","pcalign":4,"pic":tru
e,"relocs":false,"relro":"no","rpath":"NONE","static":true,"stripped":true,"su
bsys":"linux","va":true,"checksums":{}}
Build exception= canary,nx,relro
```

As of today…

- Last update of the app was January 2018
  - Update servers not reachable
  - Domain is available for re-register in March 2019

- Outdated virus scanners are a security risk
  - Helpful if PlayStore enforced update activity requirement for security tools
  - and sweep those which are orphaned (at least from being offered)

A famous device maintenance product insecure config database download

Source code of cleaner product contains http URL (CWE-311):

```
] wget http://****.****.***.***.***/tools/mobile/*****/db-v2.db,
```

which is a zip file containing a SQLite3 database, named directory.db.

```
] unzip db-v2.db

] sqlite3 directory.db
sqlite> .tables
aloneDirs       junkDirs        usefulCacheDirs excludedDirs    residualDirs
sqlite> .output clean.sql
sqlite> .dump
```

➔ Attacker could modify db and erase the directory names where he drops his stuff
➔ According to vendor should have been fixed in newest version

Hidden Gems

RANDOM FINDINGS

Inner-app version chaos

- Even within a single app multiple versions of OpenSSL are bundled
- For example in Microsoft Edge (com.microsoft.emmx) from Nov 22, 2018:
  - openssl=OpenSSL 1.1.0b 26 Sep 2016
  - openssl=OpenSSL 1.0.1u 22 Sep 2016
  - openssl=OpenSSL 1.0.2m-dev xx XXX xxxx
- Note: The vendor has been contacted multiple times since autumn 2017 about these issues in multiple packages

classes.dex: Old old Spongycastle

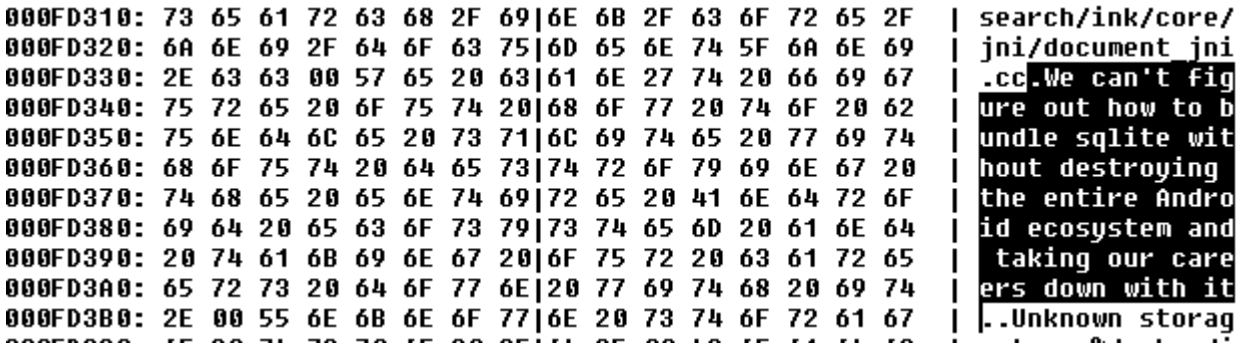
- CWE-937: Manchester United App, upgrade your old components:
  - org/bouncycastle/LICENSE.smali:17: const-string v1, "Copyright (c) 2000-2012 The Legion of the Bouncy Castle Inc. (http://www.bouncycastle.org) “
    - 6 years older than the current, https://github.com/bcgit/bc-java/blob/master/core/src/main/java/org/bouncycastle/LICENSE.java
    - Unfortunately contact address seems unmaintained
- Not as old as the MUFC version, but still three years from current, revealed by a simple grep:
  -]grep Copyright data/com.somesocialnetwork-452574.apk-unpacked/smali/org/spongycastle/LICENSE.smali
    - const-string v0, "Copyright (c) 2000-2015 The Legion of the Bouncy Castle Inc. (http://www.bouncycastle.org) “

lib: Too many revealing paths

Developer names in binaries, invitation for social engineering attacks on the build system:

```
Y*      lib\armeabi-v7a\libambisonic_audio_renderer.so: /tmp/ndk-tr****/tmp/build-72533
O*      lib\armeabi-v7a\libopera.so: /home/s*****
B*      lib\armeabi\libFFmpeg.so: /home/l**/android/ffmpeg-2.2.2/
D* lib\armeabi-v7a\libDropboxXplat.so: /Users/p****/
R*      lib\arm64-v8a\libopencv_core.so: /Users/l****/
B*      lib\armeabi-v7a\libopencv_core.so: /Users/j*****/
D*      lib\armeabi\libDRWScanSLib.so:  /tmp/ndk-andre****/tmp/build-24347
F*      lib\armeabi-v7a\libopencv_core.so: /Users/j**/
L*      lib\armeabi\libvdc.so: /usr/local/google/home/andr***
B*      (assets\avla)libavla.so: /home/d***/workspace/android-ndk-r9d/platforms/
G*      lib\armeabi\libstrongswan.so: /home/a*****/git/pwf-android/
Li*     lib\armeabi\libbrick.so: /Users/m*****/Documents/
[...]
```

lib: Quote in code



```
000FD310: 73 65 61 72 63 68 2F 69|6E 6B 2F 63 6F 72 65 2F  | search/ink/core/
000FD320: 6A 6E 69 2F 64 6F 63 75|6D 65 6E 74 5F 6A 6E 69  | jni/document_jni
000FD330: 2E 63 63 00 57 65 20 63|61 6E 27 74 20 66 69 67  | .cc.We can't fig
000FD340: 75 72 65 20 6F 75 74 20|68 6F 77 20 74 6F 20 62  | ure out how to b
000FD350: 75 6E 64 6C 65 20 73 71|6C 69 74 65 20 77 69 74  | undle sqlite wit
000FD360: 68 6F 75 74 20 64 65 73|74 72 6F 79 69 6E 67 20  | hout destroying
000FD370: 74 68 65 20 65 6E 74 69|72 65 20 41 6E 64 72 6F  | the entire Andro
000FD380: 69 64 20 65 63 6F 73 79|73 74 65 6D 20 61 6E 64  | id ecosystem and
000FD390: 20 74 61 6B 69 6E 67 20|6F 75 72 20 63 61 72 65  |  taking our care
000FD3A0: 65 72 73 20 64 6F 77 6E|20 77 69 74 68 20 69 74  | ers down with it
000FD3B0: 2E 00 55 6E 6B 6E 6F 77|6E 20 73 74 6F 72 61 67  | ..Unknown storag
```

"We can't figure out how to bundle sqlite without destroying the entire Android ecosystem and taking our careers down with it"

(Google Keep, lib\armeabi-v7a\libsketchology_native.so)

Hidden Gems

BEHIND THE SCENES: TOOLING

Scanapk

- Results were collected with a python-based tool scanapk.py
  - Recursive unpacking of APKs
  - Evaluate ELF binaries for embedded known libraries
  - Detection of version numbers and key material
  - Check imports for dangerous C functions
  - List exported JNI methods
  - List certificate information (hash algo, signer, validity)
  - List dangerous permissions
  - Uses Radare2, OpenSSL and aapt, xz, curl and „strings“
- kpa_esrever add-on performs inspection of resources.arsc, classes.dex
  - Applying rules on Dex rules to detect CWE bugs
  - Searching for key material
  - Coarse grain detection on Smali, filtering for manual inspection, optional decompilation (jadx)
  - Detect JNI entry points from Java side (Frida hooking, experimental)
- Already on GitHub, kpa_esrever addition will be updated a week after HITB

Summary

- Although critical functionality emergency apps and virus scanners have typical security weaknesses
- Privacy may be affected by tracking systems
- Warning functionality not prioritized during runtime
  - App may instead be busy with displaying ads or sending trackers
  - Advanced usability may only be available in paid version
  - Outdated embedded software may increase attack surface

Q & A

marc.schoenefeld@gmx.org
or contact on LinkedIn

https://www.reddit.com/r/ProgrammerHumor/comments/6qlpwd/the_whole_droid_family/