



<http://www.flickr.com/photos/bcostin/2619263350/>

# CVE-2013-4787 (Master Key Vulnerability) & Zip implementation of Android

Masata NISHIDA  
AVTOKY02013.5 (16th Feb. 2014)  
English Ver.

# Who am I?

- Masata Nishida (西田 雅太)
- SecureBrain
- I'm not a malware researcher, I'm just a software developer.
- Rubyist
- [@masata\\_masata](https://twitter.com/masata_masata)

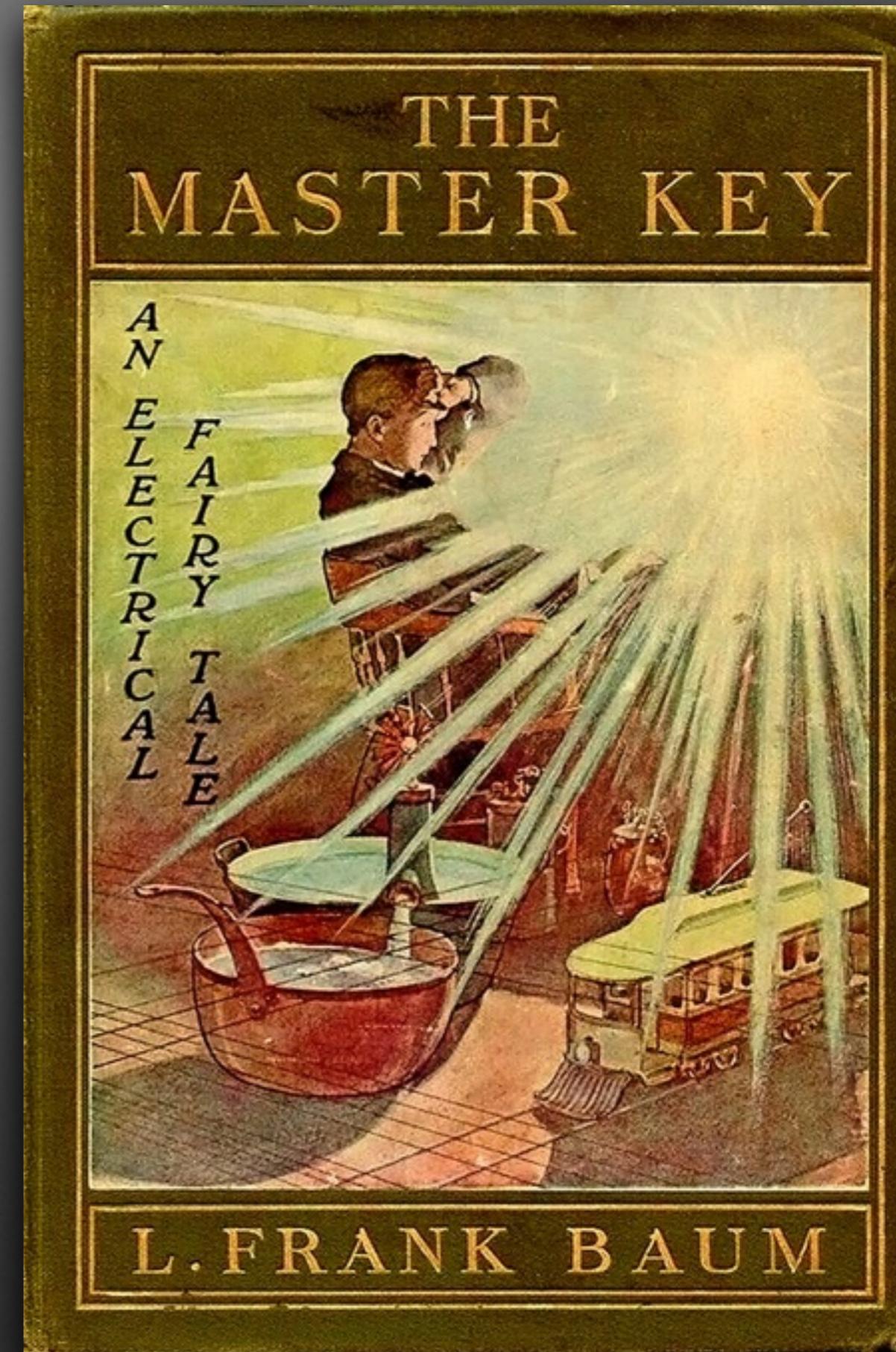


# Agenda

- Explanation about CVE-2013-4787 with source code
- ~~About Zip implementation of Android~~

# CVE-2013-4787

## Master Key Vulnerability



# CVE-2013-4787 Master Key Vulnerability

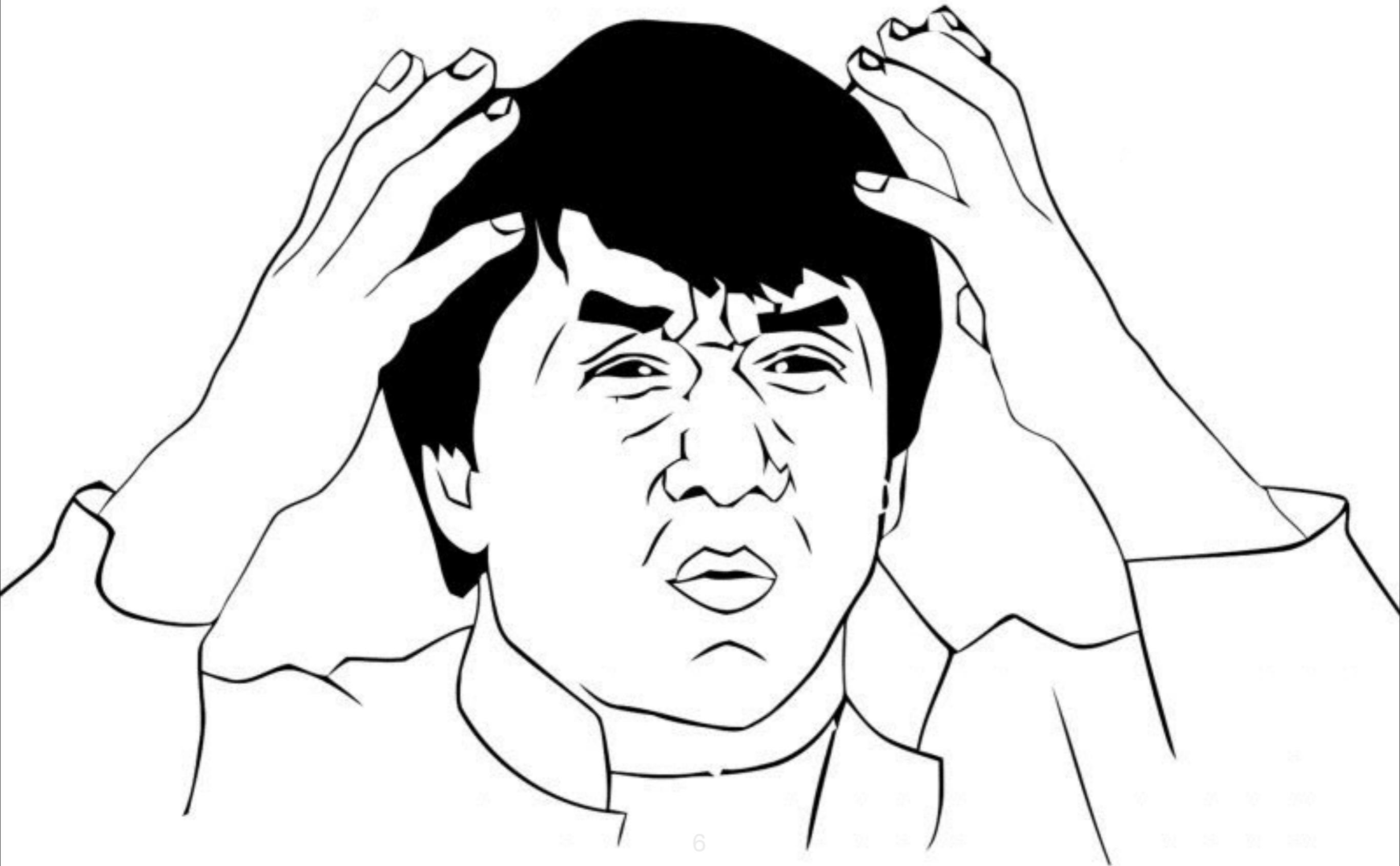
## Uncovering Android Master Key That Makes 99% of Devices Vulnerable



Written By Jeff Forristal, Bluebox CTO

- **Vulnerability in Android OS**  
The Bluebox Security research team – Bluebox Labs – recently discovered a vulnerability in Android's security model that allows a hacker to modify APK code without breaking an application's cryptographic signature, to turn any legitimate application into a malicious Trojan, completely unnoticed by the app store, the phone, or the end user. The implications are huge! The vulnerability has existed since the release of Android 1.6 ("Donut"), could affect any Android phone released in the last 4 years<sup>1</sup> – or nearly 900 million devices<sup>2</sup> – and depending on the type of application a hacker can exploit the vulnerability for anything from data theft to creation of a mobile botnet.
- **The Attacker can inject any code into the app without changing the signature of target application.**  
⇒ huge impact & user can't notice this attack.

# OH CRAP !!



# Keywords

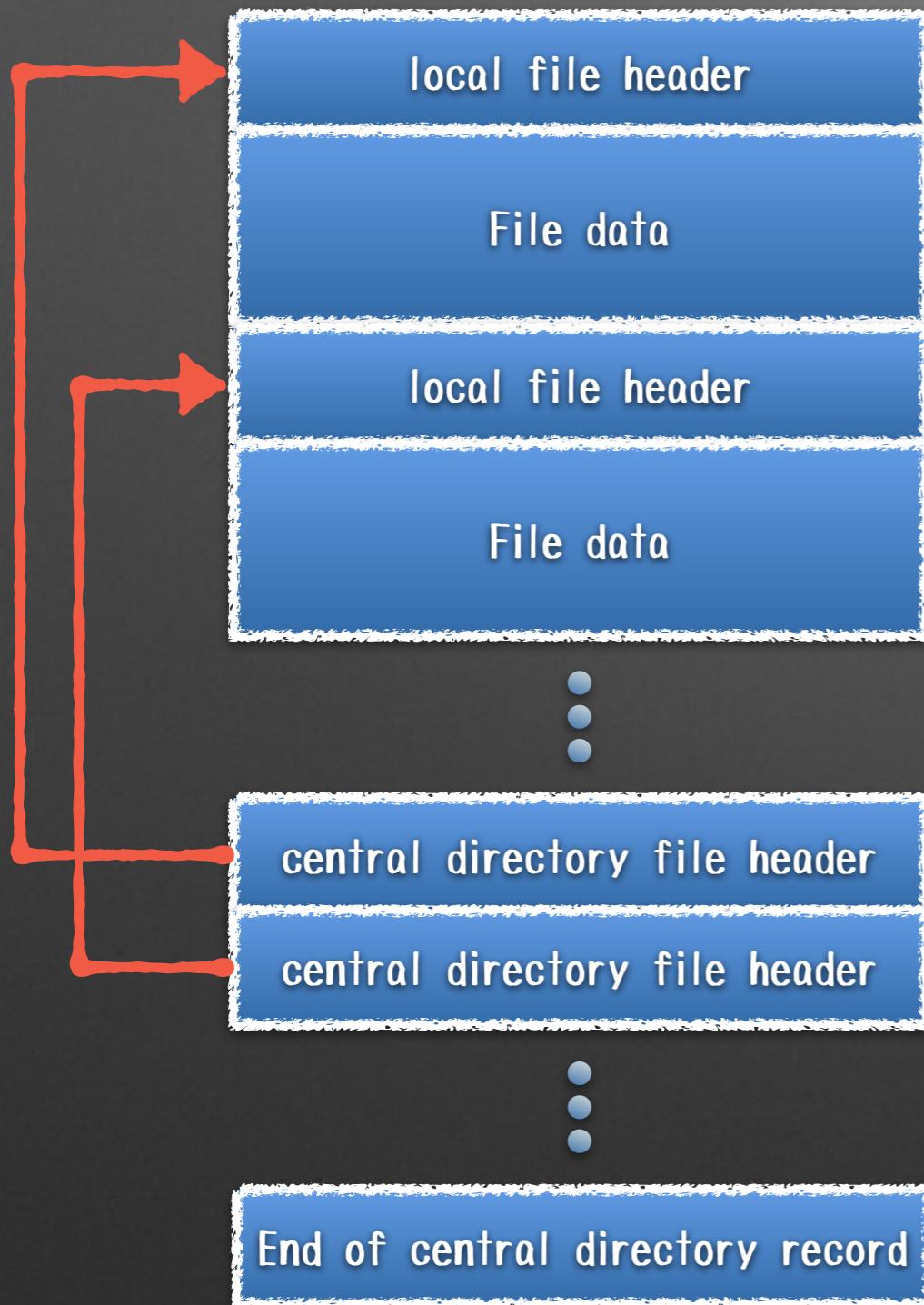
- Apk file = Zip file
- Zip file format
- Confirming a signature of Apk

# APK File (Android Application Package)

APK file is Zip file.  
It must include executable file for dalvik VM and manifest file.



# Zip Format



Local File Header (file information)  
File Data (compressed data)

Central Directory File Header  
file name, file size,  
offset of file data,  
compression method...etc

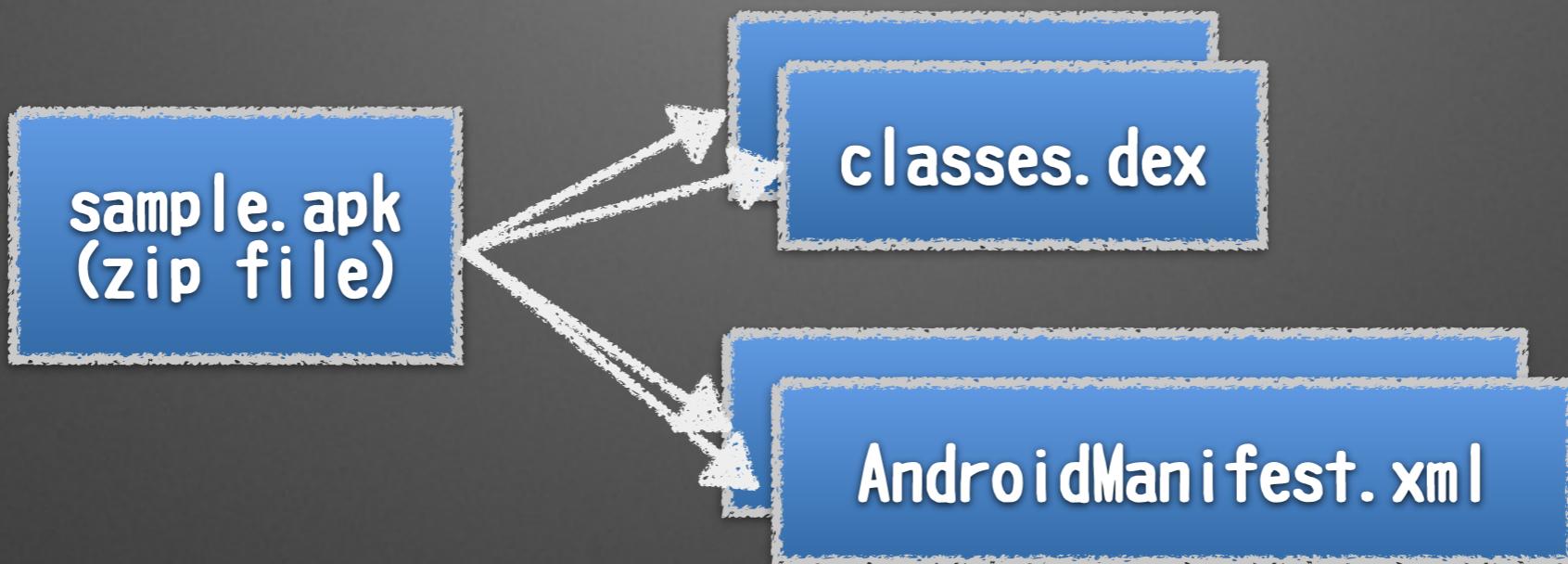
# Confirming a signature

- All applications must be digitally signed.
- You can't change any files in the apk after signing.
- Android checks the sign when user installs an new application.

If an apk has duplicated file name entries...



# APK File duplicated files



According to the specification, zip file can contain duplicated file name entries.

It's the implementation problem.

# APK File duplicated files



Inject classes.dex and manifest file into apk,  
and install it...



You can install the application  
includes another code with  
original signature!

# Why ?

- Android OS has a number of zip implementation.
  - checking signature → `java.util.zip`
  - installing application → C++
    - `frameworks/native/libs/utils/ZipFileR0.cpp`
    - `dalvik/libdex/ZipArchive.cpp`

The behavior differences between java and C++ implementation cause the issue when an Apk includes duplicate entries.

```

func isodatestring(t *time.Time) string {
    return fmt.Sprintf("%04d-%02d-%02d", t.Year(), t.Month(), t.Day)
}

func initialcode(c *http.Conn, t string, b string, e string) {
    io.WriteString(c, fmt.Sprintf(initfmt, t, canvasfill, canvasWidth,
        canvasHeight, fontname, textfill, t, canvasWidth/2, tloc, fontname))
    io.WriteString(c, legend(b+daybegin, e+dayend, tmargin, 18, 18)))
}

func legend(b string, e string, y int, w int, h int) string {
    days := int(secbetween(b, e)/secondsPerDay) + 2
    var w2 int64
    var pl = picwidth + lmargin
    var ds string
    w2 = int64(w / 2)
    yh := y - h
    ib := isosec(b)
    ie := isosec(e)
    s := ""
    lx := ib
    for i := 0; i < days; i++ {
        x = vmap(lx, ib, ie, pl, rmargin)
        //fmt.Println("lx : ", lx)
        ds = isodatestring(time.SecondsToUTC(lx))
        s = s + fmt.Sprintf(legendfmt, x, y, x-w2, yh, x+w2, yh,
            textfill, ds[0:18], x, yh-3)
        lx += secondsPerDay
    }
    s += setfont
    //fmt.Println(s)
    return s
}

func readjson(c *http.Conn, r io.ReadCloser, b string, e string, yx int) int {
    var twitter JTweets
    var data []byte
    var ntweets int
    data, err := r.Read()
    if err != nil {
        if err == nil {
            ok, errtok := json.Unmarshal(string(data), &twitter)
            if ok {
                ntweets = len(twitter.Results)
                if ntweets > 0 {
                    //fmt.Println("Ns Ns:\n",
                    //twitter.Results[0].From_user,
                    //twitter.Results[0].Profile_image_url)
                    var pl = picwidth + lmargin
                    io.WriteString(c, fmt.Sprintf(userfmt, y, y, y, y, lmargin, y,
                        picwidth, picwidth, y, twitter.Results[0].Profile_image_url, textfill,
                        twitter.Results[0].From_user, ntweets, pl+5, y+picwidth, linefill,
                        pl, y, rmargin-pl, lineHeight, markerfill)))
                    for i := 0; i < ntweets; i++ {
                        //fmt.Println("Xs Xd\n",
                        //twitter.Results[i].Created_at,
                        //rfc1123sec(twitter.Results[i].Created_at))
                        io.WriteString(c, fmt.Sprintf(pubfmt,
                            vmap(rfc1123sec(twitter.Results[i].Created_at),
                                isosec(b), isosec(e), pl, rmargin),
                            y+(lineHeight/2), markerwidth/2)))
                    }
                    return ntweets
                } else {
                    fmt.Printf("unable to parse the JSON : (%v)\n", errtok)
                }
            }
            return ntweets
        }
    }
}

func finalcode(c *http.Conn) {
    io.WriteString(c, fmt.Sprintf(endfmt, canvasWidth, canvasHeight))
}

func lf(c *http.Conn, b string, e string, n int, y int, s string) int {
    var qs string
    var ntf int = 0
    if n < 0 {
        ntf = n
    }
    if s == "" {
        qs = s
    } else {
        qs = "from:" + s
    }
}

```

Let's read source code about parsing central directory header in zip file.

# ZipFileR0 & libdex/ZipArchive

- libdex/ZipArchive is almost the same as ZipFileR0.
- It parses Central Directory File Header.  
Then it sets the file names into hash table.

```

bool ZipFileRO::parseZipArchive(void)
{
:
:
const unsigned char* ptr = cdPtr;
for (int i = 0; i < numEntries; i++) {
:
:
unsigned int fileNameLen, extraLen, commentLen, hash;

fileNameLen = get2LE(ptr + kCDENameLen);
extraLen = get2LE(ptr + kCDEExtraLen);
commentLen = get2LE(ptr + kCDECommentLen);

/* add the CDE filename to the hash table */
hash = computeHash((const char*)ptr + kCDELen, fileNameLen);
addToHash((const char*)ptr + kCDELen, fileNameLen, hash);

ptr += kCDELen + fileNameLen + extraLen + commentLen;
:
:
}
:
:
}

```

**append file name into hash table**

**compute hash value with file name**

# ZipFileR0 & libdex/ZipArchive

```
void ZipFileR0::addToFile(const char* str, int strLen, unsigned int hash)
{
    int ent = hash & (mHashTableSize-1);

    /*
     * We over-allocate the table, so we're guaranteed to find an empty slot.
     */
    while (mHashTable[ent].name != NULL)
        ent = (ent + 1) & (mHashTableSize-1);

    mHashTable[ent].name = str;
    mHashTable[ent].nameLen = strLen;
}
```

ZipFileR0 finds next empty element and sets the file name into the hash table, if the hash value is duplicated.  
→ use first item.

# Zip implementation in Java

```
public class ZipFile implements Closeable, ZipConstants {

    private final LinkedHashMap<String, ZipEntry> mEntries = new LinkedHashMap<String, ZipEntry>();

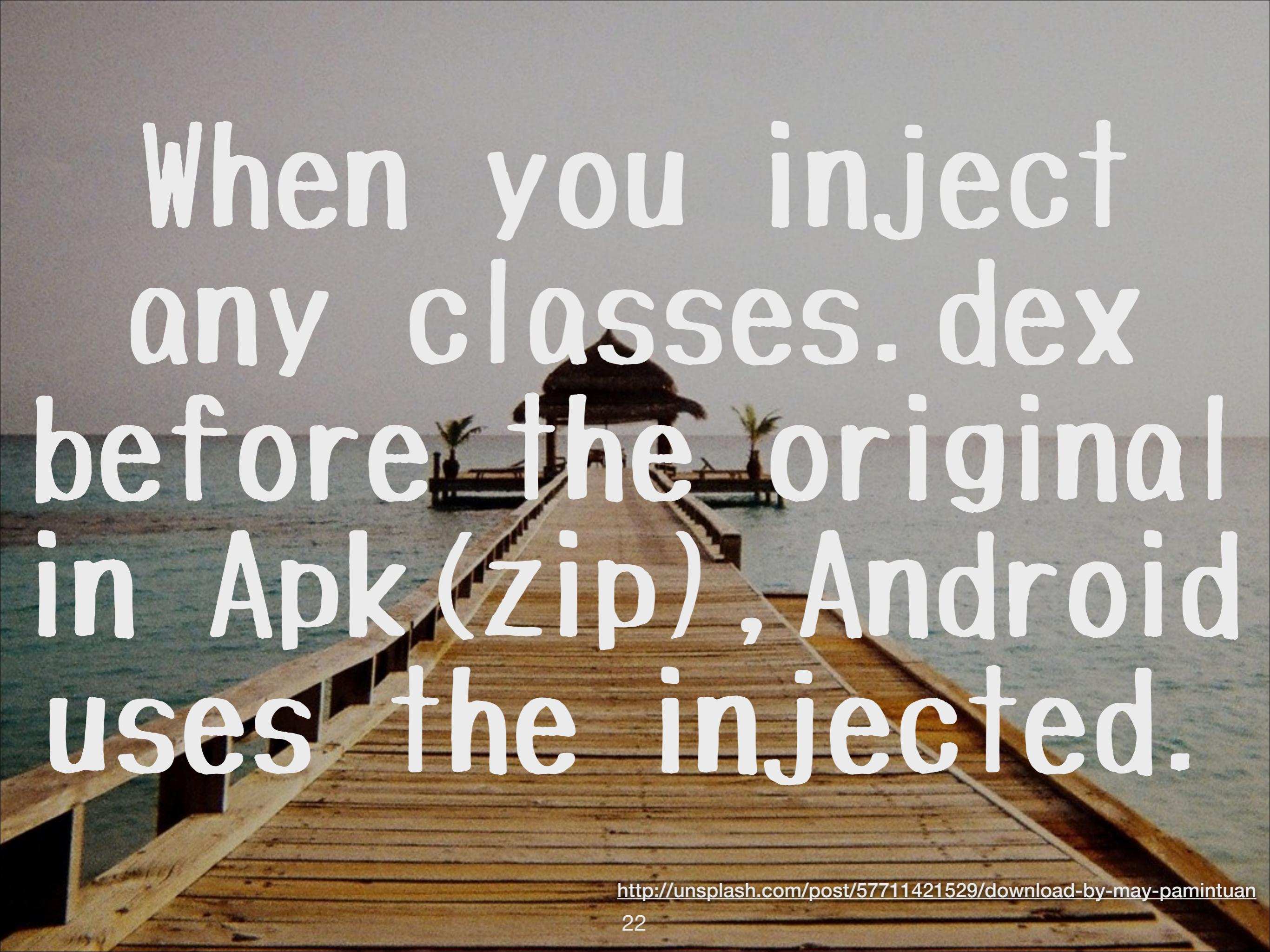
    private void readCentralDir() throws IOException {
        :
        :
        RAFStream rafStream = new RAFStream(raf, centralDirOffset);
        BufferedInputStream bufferedStream = new BufferedInputStream(rafStream, 4096);
        byte[] hdrBuf = new byte[CENHDR]; // Reuse the same buffer for each entry.
        for (int i = 0; i < numEntries; ++i) {
            ZipEntry newEntry = new ZipEntry(hdrBuf, bufferedStream);
            mEntries.put(newEntry.getName(), newEntry);
        }
    }
}
```

Java sets zip entries into `HashMap`. The file names are used for the key of `HashMap`.

duplicated file name → overwrite `HashMap` → use last item

# … therefore

- Checking signature → use last item
  - use last zip entry
- Installing application → use first item
  - use first zip entry

A photograph of a wooden pier extending from the foreground into a body of water. In the distance, a small hut with a thatched roof sits on stilts. The sky is clear and blue.

When you inject  
any classes. dex  
before the original  
in Apk(zip), Android  
uses the injected.

# Patch

Android security bug 8219321

```
// Seek to the first CDE and read all entries.  
RAFStream rafs = new RAFStream(mRaf, centralDirOffset);  
BufferedInputStream bin = new BufferedInputStream(rafs, 4096);  
byte[] hdrBuf = new byte[CENHDR]; // Reuse the same buffer for each entry.  
for (int i = 0; i < numEntries; ++i) {  
    ZipEntry newEntry = new ZipEntry(hdrBuf, bin);  
    -    mEntries.put(newEntry.getName(), newEntry);  
    +    String entryName = newEntry.getName();  
    +    if (mEntries.put(entryName, newEntry) != null) {  
    +        throw new ZipException("Duplicate entry name: " + entryName);  
    +    }  
    }  
}
```

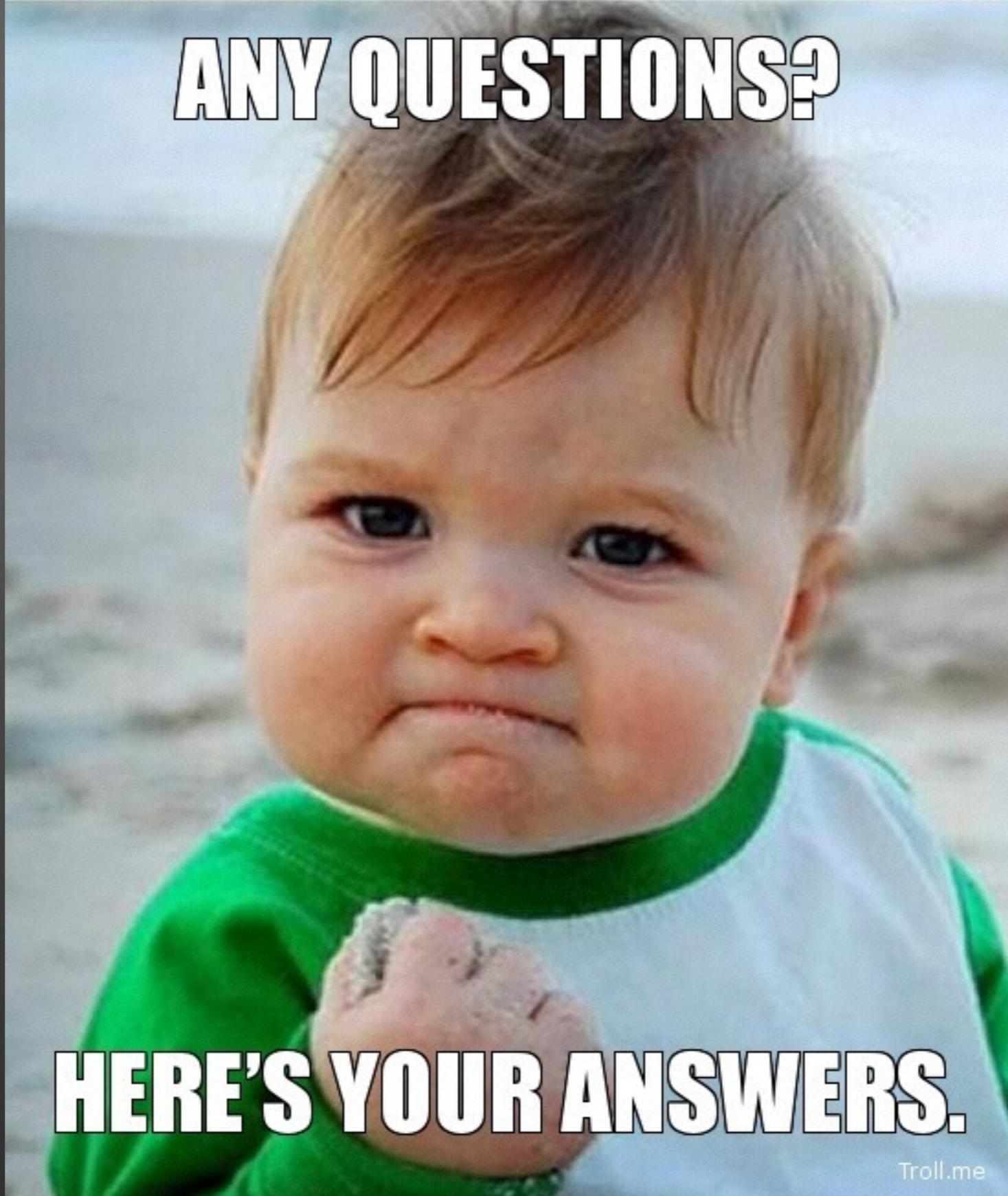
Fix Java code.  
Throw exception when duplicated entry is found.



## Appendix : Iffy zip implementation in Android

# Zip implementation in Android

- Android doesn't check zip file header in detail when it installs an application.
- Ignoring encrypt flag.
- Only deflate and no compressed are allowed as compression method.

A close-up photograph of a baby with light brown hair and blue eyes. The baby has a neutral, slightly confused or unimpressed expression. They are wearing a green and white striped shirt and are holding a pink, crumpled piece of paper or cloth in their hands.

**ANY QUESTIONS?**

**HERE'S YOUR ANSWERS.**

Troll.me