

Hashdays 2012 Android Challenge

Axelle Apvrille, Fortinet

November 2-3, 2012

Abstract

This is the solution to the Android challenge released at Hashdays 2012 [has12], in Lucerne, Switzerland, November 2-3 2012.

If you are willing to try the challenge, stop reading this document as it spoils it all!

1 Challenge

The challenge consists in a single APK file, with no particular additional explanation.

```
-rw-r--r-- 1 axelle axelle 46627 Sep 21 12:00 hashdays-challenge.apk
```

If you want to check your version:

```
sha1 : 0b12fd28a2d912762d37379e69189cd427eb8bbc
sha256: 8acfac2d1646b7689e09aab629a58ba66029b295068ca76cdaccbdc92b4e5ea9
```

2 First glance

The .apk extension is for Android Packages, and it looks like this package is indeed an Android package: it uses the ZIP format and note the presence of the typical classes.dex (Dalvik Executable) file.

```
$ file hashdays-challenge.apk
hashdays-challenge.apk: Zip archive data, at least v2.0 to extract
$ unzip -l hashdays-challenge.apk
Archive:  hashdays-challenge.apk
  Length      Date    Time    Name
-----
    651  2012-09-21  11:01  META-INF/MANIFEST.MF
    772  2012-09-21  11:01  META-INF/FORTIGUA.SF
   1373  2012-09-21  11:01  META-INF/FORTIGUA.RSA
   1352  2012-09-21  10:16  AndroidManifest.xml
   1920  2012-09-21  10:12  resources.arsc
  14984  2012-09-21  11:01  classes.dex
  27232  2012-09-18  16:18  assets/blob.bin
   2930  2012-09-11  17:43  res/drawable-hdpi/logo.png
   1846  2012-09-11  17:43  res/drawable-ldpi/logo.png
   2003  2012-09-11  17:43  res/drawable-mdpi/logo.png
   1120  2012-09-21  10:16  res/layout/main.xml
-----
  56183
                   11 files
```

Let's run it in an Android Emulator (Figures 1 and 2):

```
$ ./adb install /tmp/hashdays-challenge.apk
2499 KB/s (46627 bytes in 0.018s)
    pkg: /data/local/tmp/hashdays-challenge.apk
Success
```

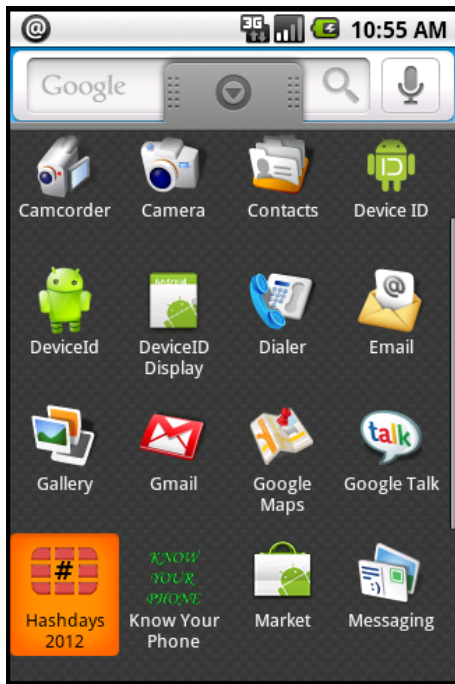


Figure 1: A new challenge application is installed on the device

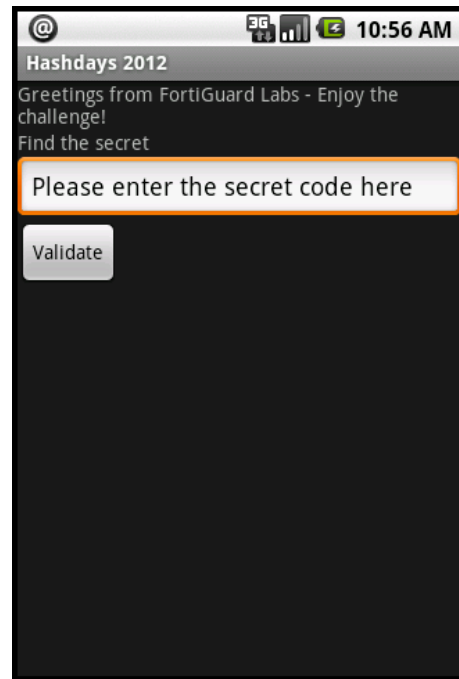


Figure 2: Find the password for fame, fun and no profit :)

So, as usual, this is all about finding the right password.

3 Reverse engineering

3.1 Disassembling

We decompile the challenge with **apktool** [apk].

```
$ java -jar apktool.jar d hashdays-challenge.apk ./apktool-output
I: Baksmaling...
I: Loading resource table...
I: Loaded.
I: Loading resource table from file: /home/axelle/apktool/framework/1.apk
I: Loaded.
I: Decoding file-resources...
I: Decoding values*/*.XMLs...
I: Done.
I: Copying assets and libs...
```

The reversed code is in `./apktool-output/smali`. If we are more familiar with Java code, we could use *dex2jar* [Dexb].

3.2 Android Manifest

The AndroidManifest is extremely simple with a single activity, named PuzzleActivity, which is started when the application is launched.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest android:versionCode="1" android:versionName="1.0"
3     package="com.fortiguard.challenge.hashdays2012.challengeapp"
4     xmlns:android="http://schemas.android.com/apk/res/android">
5     <application android:label="@string/app_name" android:icon="@drawable/logo">
6         <activity android:label="@string/app_name" android:name="PuzzleActivity">
7             <intent-filter>
8                 <action android:name="android.intent.action.MAIN" />
9                 <category android:name="android.intent.category.LAUNCHER" />
10            </intent-filter>
11        </activity>
12    </application>
13 </manifest>
```

3.3 Nothing to see :(

PuzzleActivity is a very simple activity that displays a screen like Figure 2. When we press the Validate button, this triggers the checkSecret method of Validate:

```
invoke-static {v1}, Lcom/fortiguard/challenge/hashdays2012/
    challengeapp/Validate;->checkSecret (Ljava/lang/String;) Ljava/lang/String;
```

If you don't enjoy smali more than that jump to 3.3.2

3.3.1 The Validate class, in smali

The Validate class starts with the definition of a few static fields. The static array *hashes* is filled with hexadecimal strings:

```
1 const-string v1, "d09e1fe7c97238c68e4be7b3cd..."
2 aput-object v1, v0, v5
3 const-string v1, "4813494d137e1631bba301d5..."
4 aput-object v1, v0, v6
5 sput-object v0, Lcom/fortiguard/challenge/hashdays2012/
6     challengeapp/Validate;->hashes:[Ljava/lang/String;
```

Another static array, *answers*, is filled with messages:

```
Congrats from the FortiGuard team :)
Nice try, but that would be too easy
Ha! Ha! FortiGuard grin ;)
Are you implying we are n00bs?
Come on, this is a DEFCON conference!
```

If we have tried a few random passwords in the challenge application, we know those messages are what the application displays (see Figure 3). We probably need to get this: "Congrats from the FortiGuard team :)".

After this, a big array of continuous values (00, 01, 02... FF) are written to the String array *hexArray*. We guess this is probably for hex to string conversion. Strange, the array is not used. Perhaps a remnant of debugging.

```
$ grep -r hexArray ./smali
[...]/Validate.smali:.field public static hexArray:[Ljava/lang/String;
[...]/Validate.smali:    sput-object v0, Lcom/fortiguard/challenge/hashdays2012/
challengeapp/Validate;->hexArray:[Ljava/lang/String;
```

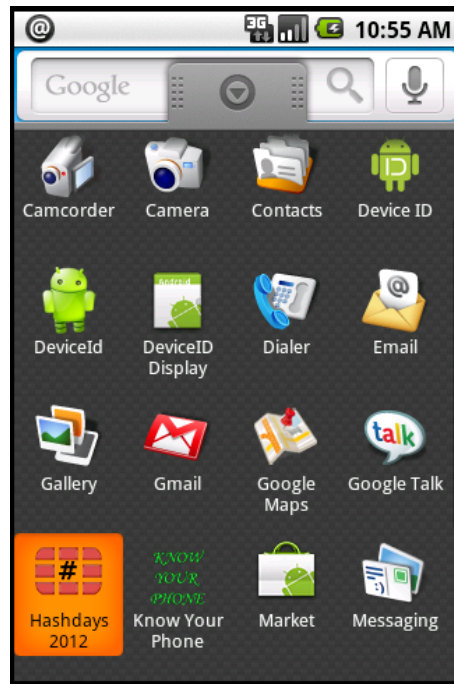


Figure 3: Bad password for the challenge

Then a multi-dimension array named *bh* is allocated, and a boolean field named *computed* is initialized to false.

The constructor does not do anything interesting apart copying the Context object, which is pretty common in Android apps.

Now, we want to dig into the `checkSecret()` method. This method takes as input parameter the password we entered, and it returns a string among *answers*. How? well obviously, it computes a sha-256 hash of the password, and checks if it matches a given value.

This part computes the hash:

This part checks the result against given values:

```

1 // the result is stored in a variable named computedHash
2 .local v0, computedHash:[B
3 ..
4 // initialize int i = 0
5     const/4 v3, 0x0
6     .local v3, i:I
7     ..
8 // compare with bh[i]
9 sget-object v4, Lcom/fortiguard/challenge/hashdays2012/challengeapp/Validate; ->bh: [[B
10 aget-object v4, v4, v3
11 invoke-static {v0, v4}, Ljava/util/Arrays; ->equals ([B[B) Z
12 move-result v4

```

What do we have in array *bh*? Actually, *bh* is filled by method `convert2bytes()`. The array hashes contains hashes as a hexadecimal string. `Convert2bytes()` writes each hash as a byte array into an entry of *bh*.

If we come back to `checkSecret()`, it checks the input's hash against a fixed set of hashes. Depending on the hash that matches, it outputs a given message from the answer array:

```
sget-object v4, Lcom/fortiguard/challenge/hashdays2012/challengeapp
/Validate;->answers:[Ljava/lang/String;
```

and if nothing matches, it returns a default string (the fourth):

```
sget-object v4, Lcom/fortiguard/challenge/hashdays2012/challengeapp
/Validate;->answers:[Ljava/lang/String;
const/4 v5, 0x4
aget-object v4, v4, v5
```

That's all. And after that, we have the `convert2bytes()` method that we already detailed, a `hexStringToByteArray()` method which obviously convert a hex string to a byte [] and a `isEmulator()` method which is particularly short:

```
1 .method public static isEmulator()Z
2     .locals 1
3
4     .prologue
5     .line 100
6     const/4 v0, 0x1
7
8     return v0
9 .end method
```

3.3.2 The Validate class, in Java

With `dex2jar`, you can read decompiled code for the `Validate` class.

Below, we will only insist on the `checkSecret()` method:

- it hashes the input with SHA-256
- checks the result against fixed values stored in the `bh` array
- depending on the match, it returns a given message in the answer array. If no hash match, it returns `answer[4]`.

```

1 public static String checkSecret(String paramString)
2 {
3     try
4     {
5         MessageDigest localMessageDigest = MessageDigest.getInstance("SHA-256");
6         localMessageDigest.reset();
7         byte[] arrayOfByte = localMessageDigest.digest(paramString.getBytes());
8         if (!computed)
9         {
10             convert2bytes();
11             break label114;
12             while (i < hashes.length)
13             {
14                 if (Arrays.equals(arrayOfByte, bh[i]))
15                 {
16                     str = answers[i];
17                     return str;
18                 }
19                 i++;
20             }
21         }
22     }
23     catch (Exception localException)
24     {
25         while (true)
26         {
27             Log.w("Hashdays", "checkSecret: " + localException.toString());
28             String str = answers[4];
29             continue;
30             label114: int i = 0;
31         }
32     }
33 }

```

3.4 Intermediate conclusion

So it seems that the application validates the input by hashing it with SHA-256 and checking the result against fixed values. The winner hash, which displays "Congrats from the FortiGuard team :)", is

622a751d6d12b46ad74049cf50f2578b871ca9e9447a98b06c21a44604cab0b4.

Of course, it is worth trying a few passwords, but you'd need to be very lucky. And until the SHA-3 competition is finished, SHA-256 is one of the most secure hashes... so good luck :(

We however notice a few strange things:

- the static array *hexArray* is not used. The *context* provided to the *Validate* constructor is not used either.
- the method *isEmulator()* is useless
- there is an asset named *blob.bin* in the package (see 2).

Let's have a look at this asset:

```
$ file blob.bin
```

```
blob.bin: data
$ hexdump -C blob.bin | head -n 3
00000000  52 e8 1d 23 fa 3d f4 d9 61 55 9c db e5 13 c6 e4 |R..#.=..aU.....|
00000010  91 36 24 9c 1a be 14 15 7e 8a 7d 01 09 b1 a9 78 |.6$......~.}....x|
00000020  03 2c b5 19 59 7f b5 81 59 ee b6 47 58 93 7e 6a |.,...Y...Y..GX.~j|
$ strings blob.bin
yNdu
Q!ym
oF' <
..
```

Really, we can't make any sense about this.

There is nothing either in the resource directory, apart our nice icon.

So what?! Well, actually, I designed the challenge, so I know the solution, I just merely hope at some point you wondered what this was all about ;)

Let's list the strings in the challenge's classes.dex:

```
$ strings classes.dex
!D5C1
@301c4cd0097640bdbfe766b55924c0d5c5cc28b9f2bdab510e4eb7c442ca0c66
@4813494d137e1631bba301d5acab6e7bb7aa74ce1185d456565ef51d737677b2
..
AES/CBC/PKCS5Padding
```

AES? Wait! We did not see anybody using AES in our code?

```
$ grep -r AES ./smali
```

Let's continue with the strings:

```
Are you implying we are n00bs?
Bad SHA-256 for
```

We did not see "Bad SHA-256 for" either... If we parse the strings more there a couple more we never saw: cutlen, doFinal, getAssets, getExternalStorageDirectory, mixed, skeySpec, whatever, work.

Something is hidden is the DEX file!

4 Hide and Seek in the DEX file

There are probably plenty of other ways to do it, but 010 Editor [010] with a template for DEX files [Lar] is extremely handy for the investigation.

We get the classes.dex from the package, load it in 010 Editor and run the DEX template on it. In the array of method_ids, we notice a method we had never seen: Validate.work() - see Figure 4.



Figure 4: Our smali code hadn't mentioned Validate.work()!

It is also possible to spot the hidden method with Androlyze [Des]:

```
In [1]: d, dx = AnalyzeDex( 'classes.dex' )

In [2]: d.methods.show()
...
34##### Method Id Item
class_idx=25 proto_idx=20 name_idx=467
class_idx_value=Lcom/fortiguard/challenge/hashdays2012/
challengeapp/Validate; proto_idx_value=['(',')', 'V'] name_idx_value=work
```

In the class_def of Validate, we however do not see that work() method - see Figure 4

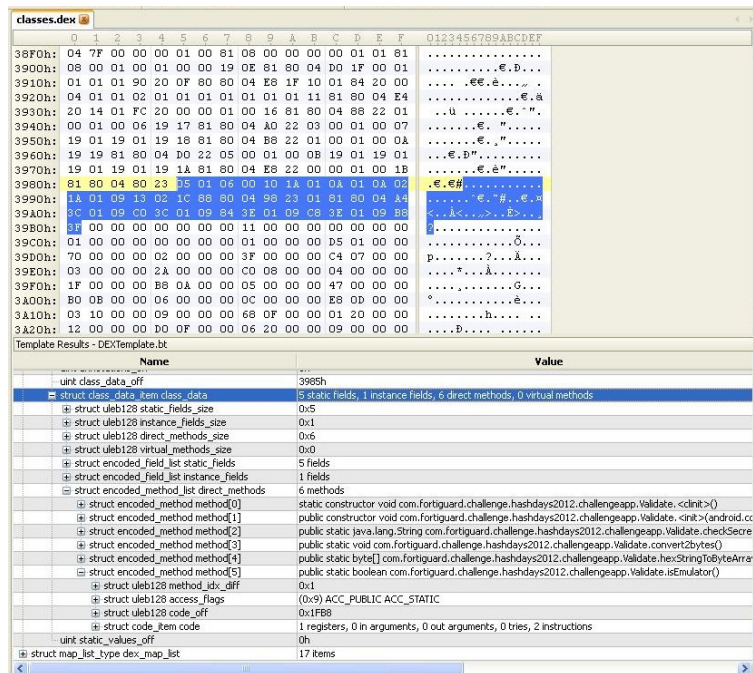


Figure 5: Validate.work() is not listed in the class definition

Additionally, there is a strange empty spot after the last method declaration in the Validate class - see Figure 6. The structure encoded_method for isEmulator() ends at 0x39b0, but the dex_map_list only starts at 0x39b8.

This is just enough for the declaration of a hidden method. To summarize: we've got unused fields (hexArray), unused strings (Bad SHA-256, AES/CBC/PKCS5Padding...), a method work() which is listed in the method list, but not in the corresponding class, an empty spot at the end of the method declaration of the Validate class, after isEmulator. Looks like the declaration for the method work() has been erased from the DEX file!

4.1 Easy way with IDA Pro

If you have IDA Pro, the easiest path to the solution is now to launch IDA Pro and look at the end of isEmulator(). What if the bytes afterward were code? (see Figure 7).

Press button P to define a function at 0x1fcc and enjoy the magic at Figure 8.

4.2 Revealing the code of the hidden method with Androguard

A slightly more difficult approach relies on the use of Androguard [Des].

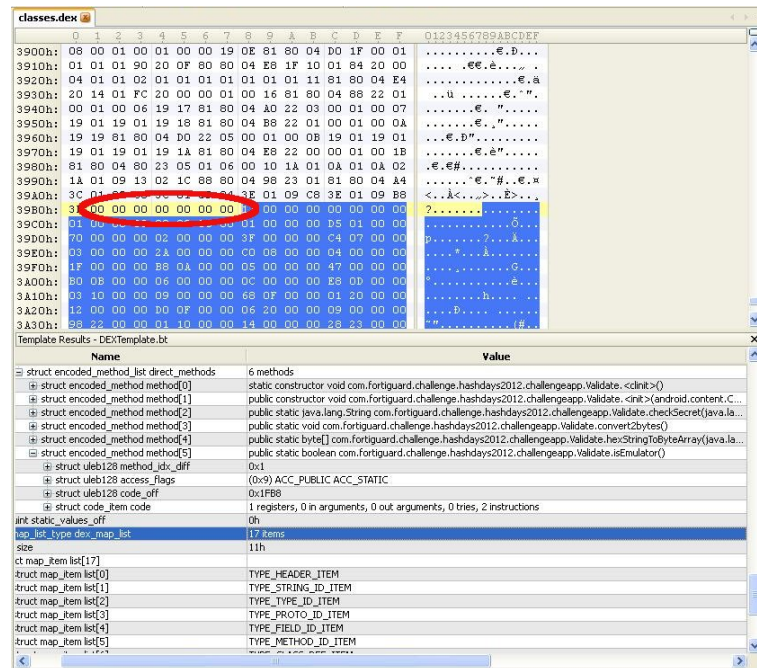


Figure 6: Null bytes at the end of declaration of method isEmulator()

As we have seen previously, our guess is that there is a hidden method after isEmulator(). Figure 9 shows isEmulator()'s code offset starts at 0x1FB8 and finishes at 0x1FCB.

So, with Androlyze, we are going to inspect isEmulator() and changes its code offset to the supposed location of the hidden method. Let's try just after 0x1FCB, i.e at 0x1FCC = 8140.

To do so, we need to patch androlyze and add a set_code.off function that specifies a new code offset.

```
1 diff -r af174939fcc1 androguard/core/bytecodes/dvm.py
2 --- a/androguard/core/bytecodes/dvm.py    Fri Sep 14 11:11:48 2012 +0200
3 +++ b/androguard/core/bytecodes/dvm.py    Tue Sep 25 16:34:58 2012 +0200
4 @@ -2684,6 +2684,10 @@ @@
5     """
6     return self.code_off
7
8 + def set_code_off(self, off) :
9 +     self.code_off = off
10 +     self.reload()
11 +
12     def get_access_flags_string(self) :
13         """
14         Return the access flags string of the method
```

Then, we use Androlyze:

```
In [1]: d, dx = AnalyzeDex( 'classes.dex' )
```

```
In [2]: isemulator = d.CLASS_Lcom_fortiguard_challenge_hashdays2012_
challengeapp_Validate.METHOD_isEmulator
```

```
In [3]: isemulator.set_code_off(8140)
```

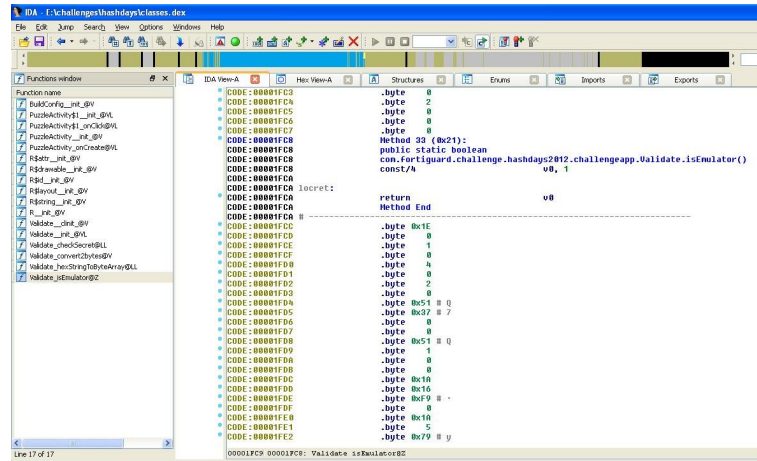


Figure 7: Dummy bytes at the end of isEmulator

```
In [4]: isemulator.get_code_off()
Out[4]: 8140
```

```
In [5]: isemulator.pretty_show()
```

The hidden method is displayed by `pretty_show()` at Figure 10.

Another way to do this without even patching Androguard is to display all the code:

```
In [1]: d = DalvikVMFormat( open('classes.dex').read() )
In [2]: codes = d.get_codes_item()
In [3]: codes.show()
```

This will display the disassembly for the all code items in the data section. To specifically show the hidden function:

```
In [1]: d = DalvikVMFormat( open('classes.dex').read() )
In [2]: d.codes.get_code(0x1fcc).show()
```

4.3 Re-constructing the method declaration

There is yet another way to solve this stage, perhaps the cleanest solution. It consists in re-constructing the missing entry for `Validate.work()`.

A method declaration consists in:

- a `method_idx_diff`. [DEXa] defines this as *”index into the `method_ids` list for the identity of this method (includes the name and descriptor), represented as a difference from the index of previous element in the list. The index of the first element in a list is represented directly”*.
- an access flag. Typically `ACC_PUBLIC = 0x1` for public access.
- offset to the code

```

.byte 0
.byte 0
Method 33 (0x21):
public static boolean
com.fortiguard.challenge.hashdays2012.challengeapp.Validate.isEmulator()
const/4 v0, 1

return v0
Method End
Method 0 (0x0):
public java.lang.String[]
android.annotation.SuppressLint.value()
monitor-exit v0
move v0, v0
move-wide v0:v1, v0:v1
move/from16 v0, v14161
nop
aput-short v1, v0, v0
const-string v22, aE6a7c9a6c6a5cc # "E6a7c9a6c6a5cc9CEADC958F405CCBFDEE2D02"...
const-string v5, aBlob_bin # "blob.bin"
move-object/from16 v0, v29
iget-object v0, v0, Validate_context
move-object/from16 v26, v0
invoke-virtual/range {v26..v26}, <ref Context.getAssets() imp. @_def_Context.getAssets@L>
move-result-object v4
const/16 v26, 0x400
move/from16 v0, v26
new-array v6, v0, <t: byte[]>
invoke-static/range {v22..v22}, <ref Validate.toHexStringToByteArray(ref) Validate_hexString
move-result-object v3
array-length v0, v3
move/from16 v26, v0
div-int/lit8 v10, v26, 2
new-array v0, v10, <t: byte[]>
move-object/from16 v21, v0
new-array v0, v10, <t: byte[]>
move-object/from16 v18, v0
const/16 v16, 0
const/16 v20, 0

```

Figure 8: The dummy bytes are Dalvik code, and they make sense :)

We want to re-construct the entry for method `Validate.work()`, just after the declaration of `Validate.isEmulator()`. After direct methods, the DEX format specifies virtual methods. So, let's declare `work()` as a virtual method. Its `method.idx_diff` will be `34 = 0x22` (see Figure 4). Its access flag will be `ACC_PUBLIC = 0x1`. Figure 9 shows `isEmulator()`'s code offset starts at `0x1FB8` and finishes at `0x1FCB`. So, we can try and set `work`'s code offset to `0x1FCC`.

We can use any hexadecimal editor to modify the DEX file at `0x39b1` and write the new entry. Beware, we are writing `uleb128` [DEXa], so "each byte has its most significant bit set except for the final byte in the sequence, which has its most significant bit clear". This means we actually write: `0x22 0x01 0xcc 0x3f`.

Then, we also need to edit the number of virtual methods of the class (address `0x3981`) and set it to 1. Finally, we need to re-compute the SHA1 and the checksum of the file.

The checksum is an Adler checksum. It is computed on all fields except magic and itself.

```

1 sub compute_dex_checksum {
2     my $filename = shift;
3     open( FILE, $filename ) or die "sha1: cant open $filename: $!";
4     binmode FILE;
5
6     # skip magic and checksum
7     get_magic(\*FILE);
8     $dex->{checksum} = get_checksum(\*FILE);
9
10    my $a32 = Digest::Adler32->new;
11    $a32->addfile(\*FILE);
12    close(FILE);
13    my $checksum = $a32->hexdigest;
14
15    return $checksum;
16 }

```

The SHA-1 hash of the file is computed on the entire file except magic, checksum and itself.

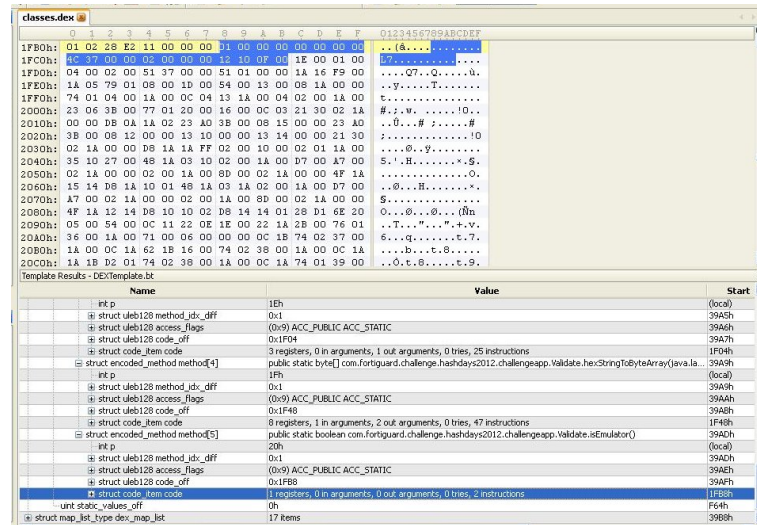


Figure 9: Code of isEmulator starts at 0x1fb8 and ends at 0x1fcb (included)

```

1  sub compute_dex_shal {
2      my $filename = shift;
3      open( FILE, $filename ) or die "shal: cant open $filename: $!";
4      binmode FILE;
5
6      # skip magic, checksum, shal
7      $dex->{magic} = get_magic(\*FILE);
8      $dex->{checksum} = get_checksum(\*FILE);
9      $dex->{shal} = get_shal(\*FILE);
10
11     # compute shal
12     my $shaobj = new Digest::SHA("1");
13     my $shal = $shaobj->addfile(*FILE)->hexdigest;
14     close( FILE );
15
16     return $shal;
17 }

```

Then, we can decompile the new DEX and, for instance, obtain the following code:

```

cess_flags=public static]
##### Params
local registers: v0...v29
- return:boolean
#####
*****
isEmulator-BB80x0 :
0 (00000000) const-string      v22, 'E6A7C9A6C6A5CCE9CEADC958F405CCB
FDEE2D023C613CB76CCEDC2C7D5BSADAE'
1 (00000004) const-string      v5, 'blob.bin'
2 (00000008) move-object/from16  v0, v29
3 (0000000c) iget-object        v0, v0, Lcom/fortiguard/challenge/has
hdays2012/challengeapp/Validate;->context Landroid/content/Context;
4 (00000010) move-object/from16  v26, v0
5 (00000014) invoke-virtual/range v26, Landroid/content/Context;->getAs
sets()Landroid/content/res/AssetManager;
6 (0000001a) move-result-object  v4
7 (0000001c) const/16           v26, #1024
8 (00000020) move/from16        v0, v26
9 (00000024) new-array          v6, v0, [B
10 (00000028) invoke-static/range v22, Lcom/fortiguard/challenge/hashda
ys2012/challengeapp/Validate;->hexStringToByteArray(Ljava/lang/String;)[B
11 (0000002e) move-result-object  v3
12 (00000030) array-length      v0, v3
13 (00000032) move/from16        v26, v0
14 (00000036) div-int/lit8       v10, v26, #+2
15 (0000003a) new-array          v0, v10, [B

```

Figure 10: A patched Androlyze reveals the hidden method

```

1 public void work()
2 {
3     AssetManager localAssetManager = this.context.getAssets();
4     byte[] arrayOfByte1 = new byte[1024];
5     byte[] arrayOfByte2 = hexStringToByteArray("E6A7C9A6C6A5CCE9CEADC958F405CCBF...");
6     int i = arrayOfByte2.length / 2;
7     byte[] arrayOfByte3 = new byte[i];
8     byte[] arrayOfByte4 = new byte[i];
9     int j = 0;
10    for (int k = 0; ; k++)
11    {
12        int m = -1 + arrayOfByte2.length;
13        if (j >= m)
14            break;
15        arrayOfByte3[k] = (byte) (0xA7 ^ arrayOfByte2[j]);
16        arrayOfByte4[k] = (byte) (0xA7 ^ arrayOfByte2[(j + 1)]);
17        j += 2;
18    }
19    InputStream localInputStream;
20    FileOutputStream localFileOutputStream;
21    Cipher localCipher;
22    try
23    {
24        localInputStream = localAssetManager.open("blob.bin");
25        File localFile = new File(Environment.getExternalStorageDirectory()
26            + File.separator + "whatever");
27        localFileOutputStream = new FileOutputStream(localFile);
28        SecretKeySpec localSecretKeySpec = new SecretKeySpec(arrayOfByte3, "AES");
29        IvParameterSpec localIvParameterSpec = new IvParameterSpec(arrayOfByte4);
30        localCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
31        localCipher.init(2, localSecretKeySpec, localIvParameterSpec);
32        MessageDigest localMessageDigest = MessageDigest.getInstance("SHA-256");
33        localMessageDigest.reset();
34        for (int n = localInputStream.read(arrayOfByte1); n != -1;
35            n = localInputStream.read(arrayOfByte1))
36        {
37            byte[] arrayOfByte7 = localCipher.update(arrayOfByte1, 0, n);
38            localMessageDigest.update(arrayOfByte1, 0, n);
39            if (arrayOfByte7 == null)
40                continue;
41            localFileOutputStream.write(arrayOfByte7);
42            localFileOutputStream.flush();
43        }
44    }
45    }

```

```

1  byte[] arrayOfByte5 = localMessageDigest.digest();
2  StringBuffer localStringBuffer = new StringBuffer();
3  for (int i1 = 0; ; i1++)
4  {
5      int i2 = arrayOfByte5.length;
6      if (i1 >= i2)
7          break;
8      localStringBuffer.append(hexArray[(0xFF & arrayOfByte5[i1])]);
9  }
10  if (localStringBuffer.toString().compareToIgnoreCase("528f83083b5df4fee...")) {
11      throw new IOException("Bad SHA-256 for " + "blob.bin");
12  }
13  catch (Exception localException)
14  {
15      Log.e("Hashdays", localException.toString());
16  }
17  while (true)
18  {
19      return;
20      byte[] arrayOfByte6 = localCipher.doFinal();
21      if (arrayOfByte6 != null)
22      {
23          localFileOutputStream.write(arrayOfByte6);
24          localFileOutputStream.flush();
25      }
26      localFileOutputStream.close();
27      localInputStream.close();
28  }
29  }

```

5 Reversing the hidden method

The code shows that:

- we load a fixed hexadecimal string `E6A7C...` and make 2 byte arrays out of it. Bytes are read 2 by 2: the first byte goes in the first array, the second to the second array. Each byte is XORed with 0xa7.
- we open the asset we had noticed, `blob.bin`, and are going to decrypt it in a file named *whatever*. The encryption algorithm is AES-CBC using PKCS5 padding. The first array of byte above is the key and the second is the initialization vector.
- we also perform a SHA-256 hash of `blob.bin`, and convert the resulting byte array to a hexadecimal string using `hexArray`. We compare the string to a fixed hash `528f8...`. If the hash does not match, we display "Bad SHA-256 for blob.bin".

We now need to decrypt `blob.bin`. There are two solutions: modify the challenge's application to add a call to `work()` and hence decrypt `blob.bin`, or write our own standalone code that decrypts `blob.bin`. The code is not difficult to write:

```

1 public static void likeAndroid() throws Exception {
2     String mixed = "E6A7C9A6C6A5CCE9CEADC958F40..";
3     InputStream is = new FileInputStream("blob.bin");
4     OutputStream os = new FileOutputStream("decrypted-out");
5
6     // decode keys
7     byte [] all = DecryptBlob.hexStringToByteArray(mixed);
8     int cutlen = all.length / 2;
9     byte [] key = new byte [cutlen];
10    byte [] iv = new byte [cutlen];
11
12    for (int i=0,j=0; i<all.length-1; i+=2, j++) {
13        key[j]=(byte) (all[i] ^ 0xa7);
14        iv[j]=(byte) (all[i+1] ^ 0xa7);
15    }
16
17    // decrypt file
18    SecretKeySpec skeySpec = new SecretKeySpec(key, "AES");
19    IvParameterSpec ivspec = new IvParameterSpec(iv);
20
21    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
22    cipher.init(Cipher.DECRYPT_MODE, skeySpec, ivspec);
23
24    byte[] b = new byte[1024];
25    int bytesRead = is.read(b);
26    while (bytesRead != -1) {
27        byte [] c = cipher.update(b, 0, bytesRead);
28        if (c != null) {
29            os.write(c);
30            os.flush();
31        }
32        bytesRead = is.read(b);
33    }
34    byte [] end = cipher.doFinal();
35    if (end != null) {
36        os.write(end);
37        os.flush();
38    }
39    os.close();
40    is.close();
41 }

```

Now, we can read the decrypted blob.bin.

```

$ hexdump -C blob.bin | head
00000000  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
00000010  20 2e 2e 2e 2e 2e 2e 2e 2e 2e 2e 2e 2e 2e 2e 2e | .....|
00000020  2e 2e 2e 2e 2e 2e 2e 2e 2e 2e 2e 2e 2e 2e 2e 2e | .....|

```

This is a bit strange. Not immediately readable. But if we do:

```

$ file blob.bin
blob.bin: ASCII text

```

It clearly states this is ASCII. So let's load it in a browser: it's ASCII art - Figure 5.

We go back to the challenge application and enter MayTheF0rceB3W1thU and get the congrats - Figure 12

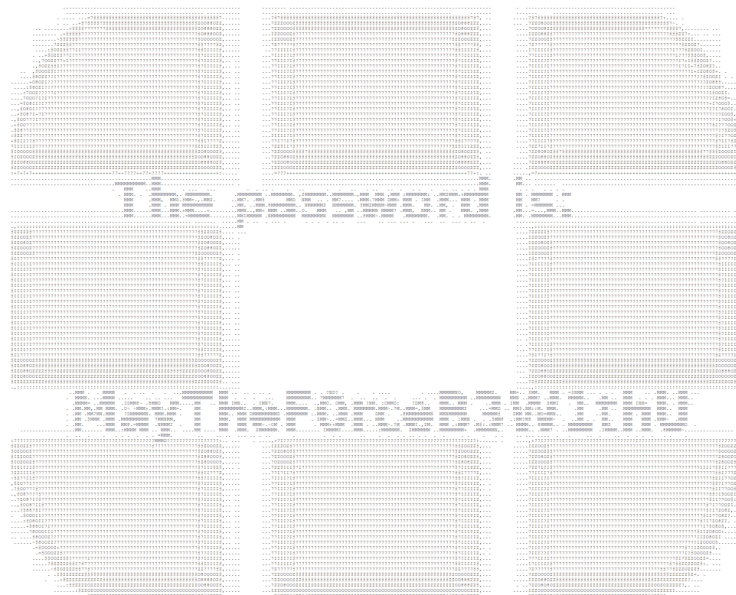


Figure 11: ASCII art for the password

6 Bonus

The AES key is `AnakinSkywalker`.

To get the message "Nice try, but that would be too easy", use `AnakinSkywalker` as password.

To get the message "Ha! Ha! FortiGuard grin ;)", use `Fortiguard` as password.

To get the message "Are you implying we are n00bs?", use `root` as password.

References

- [010] 010 Editor. <http://www.sweetscape.com/010editor/>.
- [apk] APKTool. code.google.com/p/android-apktool.
- [Des] Anthony Desnos. Androguard. http://code.google.com/p/androguard.
- [DEXa] .dex Dalvik Executable Format. <http://source.android.com/tech/dalvik/dex-format.html>.
- [Dexb] Dex2jar. <http://code.google.com/p/dex2jar/>.
- [has12] Hashdays 2012, November 2012. <https://www.hashdays.ch/>.
- [Lar] Jon Larimer. DEX Template. <https://github.com/jlarimer/android-stuff/blob/master/DEXTemplate.bt>.

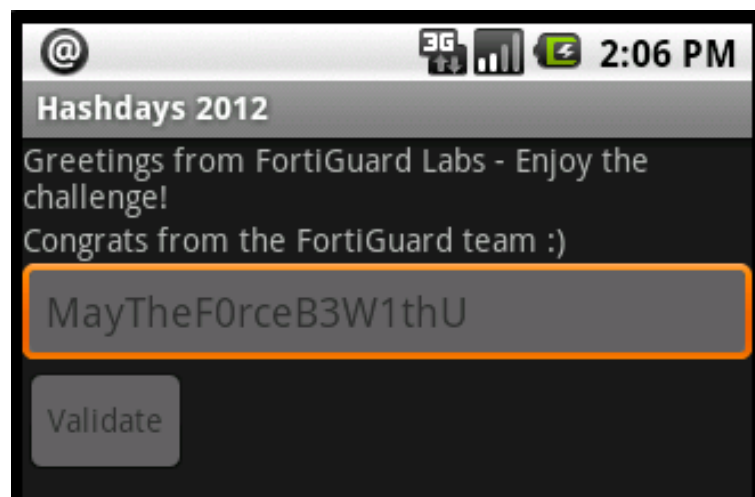


Figure 12: Congratulations for solving the challenge