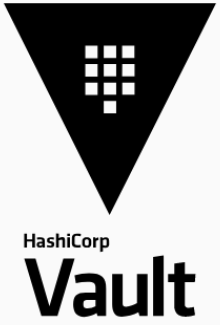
Securing Secret Keys in Android #1

Luong

Growth Session #19 - October 11-12 2018

Secret info

- Tokens
- Passwords
- Certificates
- API keys
- Other secret credentials



Why do we need to secure secret keys?

- Often your app will have secret credentials or API keys
- Could be extracted by reverse engineering
- Losing keys does not only affect the billing for the API access but may also lead to privacy issues for the users of the app.

Proguard?

- A tool to help minify, obfuscate, and optimize your code
- Reducing size as well as removing unused classes and methods
- Remove the constant names, rename classes and methods with short, meaningless names, wherever possible. Extracting the keys then takes some more time, for figuring out which string serves which purpose.

DexGuard, a commercial obfuscator, can additionally encrypt/obfuscate the strings and classes for you. Extracting the keys then takes even more time and expertise.

Embedded in resource file or constants in source code

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="api_base_url" translatable="false">https://api.redplanethotels.com/v1/%1$s/</string>
    <string name="facebook_app_id" translatable="false">4382365583227588</string>
    <string name="google_maps_api_key" translatable="false">AIzaitGsIQyMazlqvMLZitGsIQyMazlqvMLZ</string>
    <string name="google_cloud_project_id" translatable="false">243655832270</string>
    <string name="api_key" translatable="false">ok0tky3mo0an1n6kok0tky3mo0an1n6k</string>
    <string name="newrelic_api_key" translatable="false">AA9d252g4y09k6cv7x52g4y09k6cv7xy06cb35c4bc</string>
</resources>
```

```java
public class Constants {
    public  static final String FACEBOOK_APP_ID = "4382365583227588";
    public  static final String GOOGLE_MAPS_API_KEY = "AIzaitGsIQyMazlqvMLZitGsIQyMazlqvMLZ";
    public  static final String GOOGLE_CLOUD_PROJECT_ID = "243655832270";
    public  static final String API_KEY = "ok0tky3mo0an1n6kok0tky3mo0an1n6k";
    public  static final String NEWRELIC_API_KEY = "AA9d252g4y09k6cv7x52g4y09k6cv7xy06cb35c4bc";
}
```

Embedded in resource file or constants in source code

MainActivity.smali

ok0tky3mo0an1n6kok0tky3mo0an1n6k Done Replace

```
    invoke-virtual {v1, p1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/
lang/StringBuilder;

    invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

    move-result-object p1

    invoke-virtual {v0, p1}, Landroid/widget/TextView;->setText(Ljava/lang/CharSequence;)V

    :cond_0
    const-string p1, "==="

    const-string v0, 'ok0tky3mo0an1n6kok0tky3mo0an1n6k'

    invoke-static {p1, v0}, Landroid/util/Log;->i(Ljava/lang/String;Ljava/lang/String;)I

    return-void
.end method

.method public onCreateOptionsMenu(Landroid/view/Menu;)Z
    .locals 2

    invoke-virtual {p0}, Lcom/squareboat/secretkeys/MainActivity;-
>getMenuInflater()Landroid/view/MenuInflater;

    move-result-object v0
```

Embedded in resource file or constants in source code

Pros

- Easy implementation, no extra libraries or techniques
- Could be ignored in source control

Cons

- Visible in main source code and easily edited or copied.
- Easy to extract by reverse engineering (More vulnerable to decompilation of your application package)
- => No that secure

Hidden in Gradle BuildConfigs

app.properties

```
1  API_AUTHORIZATION_CODE=Basic YW5kcwdGNDXwdGwdGNDXENDXwdGNDXEXEwq
2
3  AMAZON_POOL_ID=eu-central-1:4cb98a8eb0-08fc-48c9-bvf6-e98abedf1b
4  AMAZON_BUCKET_NAME=messages
```

buildvariants.gradle

```
1  def generateBuildConfigVariables(variant) {
2      def flavorName = variant.flavorName
3      def propFile = file("../.config/${flavorName}.properties")
4      if (propFile.canRead()) {
5          Properties props = new Properties()
6          props.load(new FileInputStream(propFile))
7          if (props != null) {
8              def buildTypePostfix = variant.buildType.name == "debug" ? "_DEBUG" : ""
9              variant.buildConfigField "String", "HOST_API", "\"" + getString(props['HOST_API' + buildTypePostfix]) + "\""
10             variant.buildConfigField "String", "HOST_NAME", "\"" + getString(props['HOST_NAME' + buildTypePostfix]) + "\""
11             variant.buildConfigField "String", "XMPP_DOMAIN", "\"" + getString(props['XMPP_DOMAIN' + buildTypePostfix]) + "\""
12             variant.buildConfigField "String", "XMPP_HOST", "\"" + getString(props['XMPP_HOST' + buildTypePostfix]) + "\""
13             variant.buildConfigField "String", "XMPP_INET_HOST", "\"" + getString(props['XMPP_INET_HOST' + buildTypePostfix]) + "\""
14             variant.buildConfigField "int", "XMPP_PORT", getInt(prop['XMPP_PORT' + buildTypePostfix])
```

BuildConfig.java

Generated source files should not be edited. The changes will be lost when sources are regenerated.

```
1   /**...*/
4   package dating;
5
6   public final class BuildConfig {
7     public static final boolean DEBUG = Boolean.parseBoolean( s: "true");
8     public static final String APPLICATION_ID = "io.buddify.dating.admin.debug";
9     public static final String BUILD_TYPE = "debug";
10    public static final String FLAVOR = "admin";
11    public static final int VERSION_CODE = 75;
12    public static final String VERSION_NAME = "1.0.75";
13    // Fields from the variant
14    public static final String AMAZON_BUCKET_NAME = "messages";
15    public static final String AMAZON_POOL_ID = "eu-central-1:4cb98a8eb0-08fc-48c9-bvf6-e98abedf1b";
16    public static final String API_AUTHORIZATION_CODE = "Basic YW5kcwdGNDXwdGwdGNDXENDXwdGNDXEXEwq";
```

Hidden in Gradle BuildConfigs

Pros

- Invisible in main source code
- Simple implementation
- Could be ignored in source control

Cons

- Easy to extract by reverse engineering (More vulnerable to decompilation of your application package)
- => No that secure

Hidden in Native Libraries with NDK

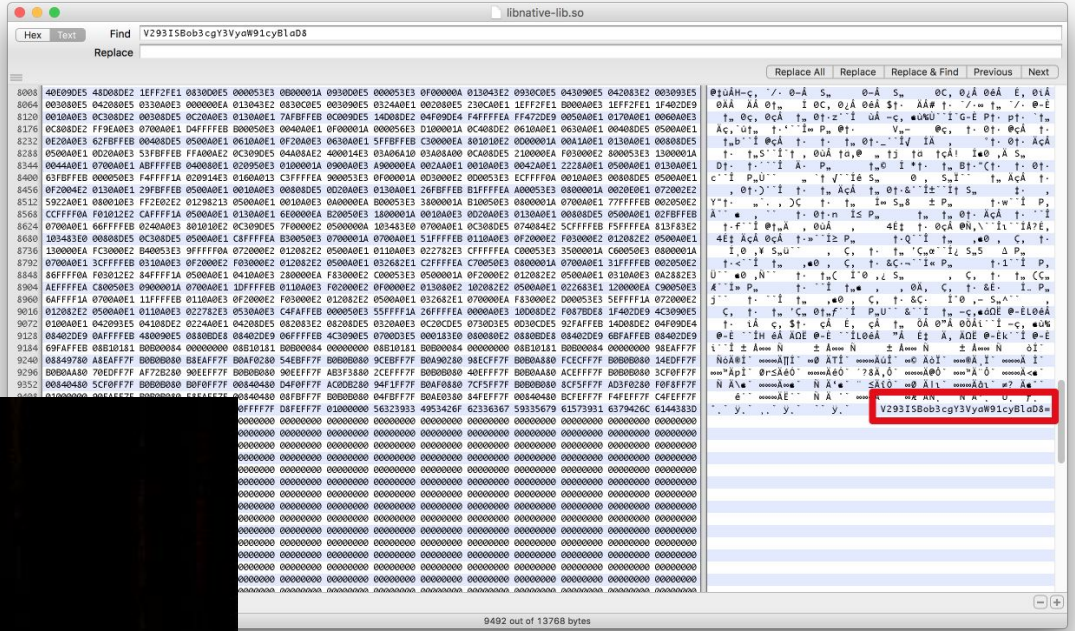
Store keys in the native C/C++ class and access them in our Java classes.

```java
static {
    System.loadLibrary("native-lib");
}

private native String invokeNativeFunction();
```

```cpp
#include <jni.h>

extern "C" {
    JNIEXPORT jstring JNICALL
    Java_info_androidsecurity_helloworld_MainActivity_invokeNat
        return env->NewStringUTF("V293ISBob3cgY3VyaW91cyBlaD8="
    }
}
```

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Context mContext = getApplicationContext();

    // Some super API call using that key
    Log.i(TAG, "key: " + invokeNativeFunction());
}
```

Hidden in Native Libraries with NDK



Hidden in Native Libraries with NDK

Pros

- More secure than Java code, can NOT be easily decompiled making the information harder to find
- A little bit harder for implementation, requires NDK knowledges

Cons

- Still can be opened with an hexadecimal editor
- => Combine with encryption
- => Convert the key into hexadecimal as it would make it less obvious when using an hex editor

Using the Android Keystore API

- Similar to the KeyChain service in iOS
- KeyStore AES API is only available from API 23
- KeyStore provides two functions:
  - Randomly generate keys
  - Securely store the keys

Using the Android Keystore API

- Generate a random key when the app runs the first time
- Store a secret:
  - Retrieve the key from KeyStore
  - Encrypt the data with it
  - Store the encrypted data in Preferences
- Read a secretL
  - Read the encrypted data from Preferences
  - Get the key from KeyStore
  - Use the key to decrypt the data

Using the Android Keystore API

Pros

- Key is randomly generated and securely managed by KeyStore and nothing except your code can read it, the secrets are secure
- Key material of Android Keystore keys is protected from extraction
- Hardware security module is supported devices running Android 9 and later
- Most secure solution

Cons

- A little tricky to implement
- Have to handle two cases: Android M (API level 23) or higher, and older Android versions
- Requires understanding of cryptography

Summary

- Absolute security does not exist.
- Combining multiple protecting measures is the key to achieving a high degree of security.
- Do NOT store String literals in your code.
- Hidden in Native Libraries with NDK is safe enough and suitable for most projects
- Using the Android Keystore API is the most secure solution and should be prefered for medium or large projects

“Absolute security does not exist. Security is a set of measures, being piled up and combined, trying to slow down the inevitable”

References

- Storing Secret Keys in Android
- Configuring ProGuard
- Storing your secure information in the NDK
- Securely Storing Secrets in an Android Application
- A follow-up on how to store tokens securely in Android

Thanks!

Contact Nimbl3

hello@nimbl3.com

399 Sukhumvit Road, Interchange 21
Klongtoey nua, Wattana
Bangkok 10110

28C Stanley St,
Singapore 068737

20th Floor, Central Tower
28 Queen's Road
Central, Hong Kong

nimbl3.com