

Radboud Universiteit Nijmegen



BACHELOR THESIS

Analysis of Android Authenticators

Author:
Raoul Estourgie

Supervisor:
Dr. ir. Erik Poll

July 8, 2013

Abstract

This thesis reports an research that has been done on the security of multi-factor authentication using Android applications. In specific I will review the Google Authenticator and the Battlenet Authenticator. There is a review on the security of the applications and also a review of the security on the server-side. This will be in multiple scenarios each focussing on a different part of the security. It appears that multi-factor authentication is very secure however the application of the Battlenet application contains a few leaks which makes it possible to obtain the cryptographic key of the application even on a non-rooted Android application.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 3 |
| 2 | Background information | 3 |
| 2.1 | One-time password algorithms | 3 |
| 2.2 | Android Operating System essentials | 5 |
| 3 | The Google Authenticator | 11 |
| 3.1 | The installation | 11 |
| 3.2 | Storing your secret key | 11 |
| 3.3 | Implementation of the protocols | 13 |
| 3.4 | Multiple authenticators for one account | 14 |
| 3.5 | Account Recovery | 14 |
| 3.6 | Application-specific passwords | 16 |
| 3.7 | Trusted Device | 18 |
| 3.8 | Generated One Time Pads on the server | 18 |
| 4 | The Battlenet Authenticator | 19 |
| 4.1 | Installation | 19 |
| 4.2 | Storing your secret key | 20 |
| 4.3 | Implementation of the protocols | 20 |
| 4.4 | Multiple authenticators for one account | 20 |
| 4.5 | Account Recovery | 21 |
| 4.6 | Trusted Device | 22 |
| 5 | Security Analysis of the Authenticators | 23 |
| 5.1 | Android backups | 23 |
| 5.2 | Debuggable android applications | 27 |
| 5.3 | Key Storage | 28 |
| 6 | Improving Security | 29 |
| 6.1 | Pin access | 29 |
| 6.2 | Android Keychain API | 29 |
| 7 | Conclusion | 31 |
| 7.1 | The Google Authenticator | 31 |
| 7.2 | The Battlenet Authenticator | 31 |
| 7.3 | Trusted Device | 31 |
| 8 | Future Work | 32 |
| 9 | Appendix | 32 |
| 9.1 | Battlenet backup file | 32 |

1 Introduction

Android is a widely used operating system for all kinds of devices. As of May 2013 more than 900 million devices have the android Operating System installed and more than 48 billion Android applications have been installed [bbc,]. This more than doubles the figures of last year. This growth also brings a lot of issue's. If more and more people keep using the Operating System it becomes more attractive for criminals to try to get access to certain data of users such as credit card details and user information. This requires the developers of the Operating System to spend extra attention to the security of the Operating System. But also the developers of the Android applications need to design a secure application that properly secures its confidential data. This thesis covers two Android applications which need to pay extra attention to the security of their data. These are the Google Authenticator and the Battlenet authenticator, two applications that are used to increase the security when logging in to a website.

2 Background information

This chapter is about the background information needed to research the application. I will delve deeper in the HMAC-based one-time password algorithm (HOTP) and Time-based One-Time Password (TOTP) algorithm. Both algorithms are quite similar but work with different counters. The battlenet authenticator only uses the TOTP algorithm, the Google authenticator uses both. I will also give a short summary about the essential parts of the Android Operating System. How applications work inside the system and how to communicate with the system.

2.1 One-time password algorithms

2.1.1 HMAC-based one-time password algorithm

RFC 4226 [MRaihi et al., 2005] defines the HMAC-based One-time Password algorithm (HOTP). This is an algorithm designed for generating One-Time password values, based on [Bellare et al., 1996] Hashed Message Authentication Code (HMAC). The HOTP algorithm is based on a "increasing counter" value and a fixed symmetric key. RFC 2104 [Krawczyk et al., 1997] describes how to generate this HMAC value.

RFC 2104 has only one requirement for the symmetric key, this is the minimum length. The length needs to be atleast 16 bytes if you want to generate an HMAC with MD5 and has a minimum requirement of 20 bytes if you want to generate your HMAC with SHA-1. The maximum length for the symmetric key is 64 bytes. Applications that use keys longer than 64 bytes will first hash the key and then use the resultant as the actual key to generate the HMAC. The HMAC-SHA-1 algorithm is recommended because its more secure then the MD5 variant.

The HMAC-SHA-1 algorithm is designed to deliver a 20 byte long code, this needs to be shortened to a more user friendly size. This is done with a truncate function reducing the 20 byte string to a 32 bit string so the code consists of at most 10 decimal characters.

The basic idea is that both the client and the server have the same key and the same counter. On the client side this can be implemented in a Java smart card, USB dongle, GSM SIM cards and a lot of other goods.

Authentication requirements

There are a certain security requirements needed to keep the attacker at bay.

- The server should not be vulnerable to brute force attacks. This means that the servers need to request an extra security option after a few failed attempts to log in. This could be done by supplying a Captcha or give a time-out.
- The protocol should be implemented over a secure channel in order to protect the users privacy.

Synchronisation of the counter The counter of the server will only increase every successful authentication, but this is not the case on the client side. Every time when the user requests a new key from the authenticator the counter will increase and the authenticator will calculate a new HOTP value. This creates the situation that the counter of the authenticator and the counter of the server do not share the same value and are out of sync.

Resynchronisation is possible by letting the server look ahead a few counter values. However you should not let the server look to far ahead, it could be a wrong combination accidentally filled in by the user or an attempt by the attacker to get the server and the client out of Synchronisation or even worse giving access to an attacker.

2.1.2 Time-based one-time password algorithm

Time-based one-time password algorithm (TOTP) [MRaihi et al., 2010] is a variation on the HOTP algorithm a time-stamp is used instead of a counter to generate values. The counter is obtained by measuring the difference between the current time and the time that is chosen as an universal starting point. The counter will then be incremented every t seconds (most of the time this is 30 seconds). The difference between the HOTP and the TOTP algorithm is that the person now has a time limit to use the one time password. It's also harder for the server and the client to get out of sync this requires the wrong time setting on one of the devices. The time requirement can of course be bothersome with a slow connection between client and server. If the one-time password reaches the server too late it can be refused because the server is expecting the one-time password matching the next time frame. In this situation it could be wise to look back a time frame and accept it anyway. The server could become vulnerable to replay attacks if it looks back more then one frame.

2.2 Android Operating System essentials

The android operating system is the dominating the tablet and the smart phone market [Alugbue,]. Android is a Linux-based operating system, designed for touchscreen mobile devices. The linux kernel is used for its device drivers, memory management, process management and networking. The next level contains the Android native libraries. This layer enables the device to handle different types of data. These libraries are written in c or c++. A small overview of these libraries are:

- Media framework
 - Provides different media codecs, this allows the recording and playback of different media devices.
- SQLite
 - The database engine used in android for data storage.
- WebKit
 - The browser engine used to display HTML content.

The next level is called the Android runtime, this consists of the Dalvik Virtual Machine and Core Java libraries. The Dalvik Virtual Machine is used to run applications, it is optimized for low processing power and low memory environments. The Dalvik Virtual Machine runs .dex files. The applications are developed in java so are .class files afterwards they are converted to .dex which provides higher efficiency in low resource environments. The Dalvik Virtual Machine provides extra security features discussed in 2.2.1. The Java libraries are the libraries you can use to develop your application together with the Application Framework.

The Application Framework is the next level in the Android Operating System. These are the block an Android application directly interacts with. These programs manage the basic functions the device has, which you can use to build the application. A small overview of a few blocks are:

- Activity Manager
 - Manages the life cycle of applications
- Content Providers
 - Manage the data sharing between applications
- Location Manager
 - Used for location management, using GPS or cell tower

The last level are the applications itself. The next page shows a visual overview of the Android Operating System:



Figure 1: The Android Operating System

2.2.1 Android security mechanisms

One of the main targets of the operating system should be security, otherwise the data of all the users would be right on the street. In [Shabtai et al., 2010] is an extensive security evaluation of the Android Operating System. The table below points out the main security aspects of the operating system.

| Mechanism | Description | Security issue |
|--|--|---|
| Linux mechanisms POSIX users File access | Every application has its own userID Each application has its own storage space | Prevents applications to access the other application Prevents applications from accessing files of other applications |
| Environmental features Type safety Mobile carrier security features | Type safety enforces variable content to adhere to a specific format, both in compile time and runtime. Smart phones use SIM cards to authenticate and authorize user identity. | Prevents buffer overflows and stack smashing Prevents phone call theft |
| Android-specific mechanisms Application permissions Application signing Dalvik Virtual Machine | Each application shows the permissions it requires before it is installed. The developer signs the application, which is then verified by the package manager Each application runs in its own virtual machine | Alerts users on applications with too many permissions for its function. This is an authenticity check Prevents buffer overflows, remote code execution and stack smashing. |

It is important to be aware of the security limits of the operating system, so we have a clear overview of the main focus of the application security itself.

2.2.2 What is an Android application exactly

Before an application is installed it is an .apk package. This package contains the following items.

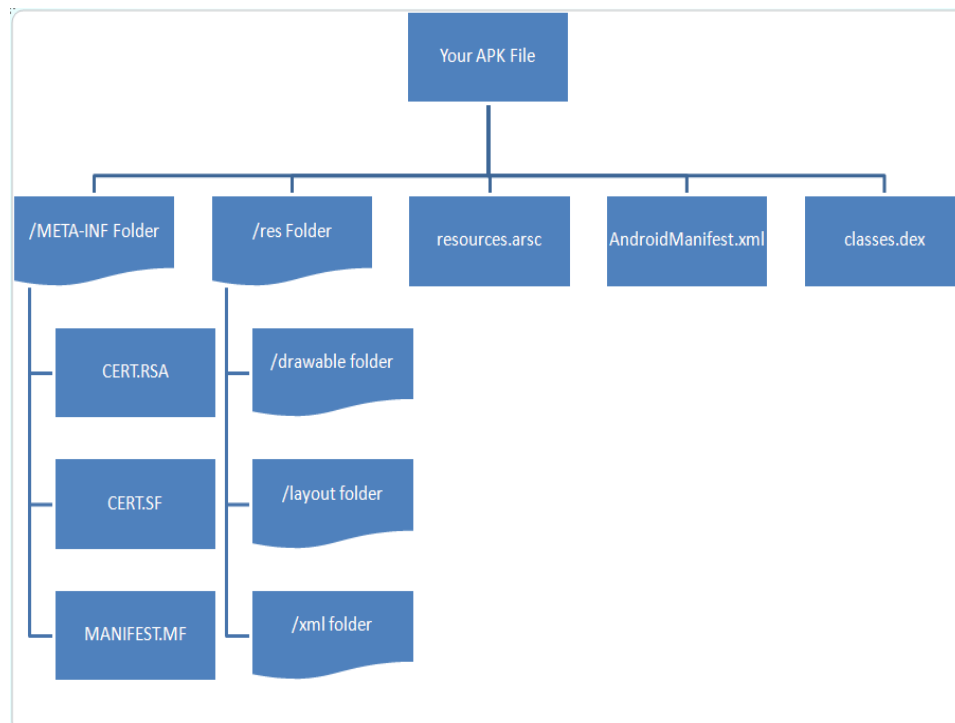


Figure 2: Overview of the file structure of an APK file

- The META-INF folder
 - Contains the certificate of the application.
 - Contains SHA1 hashes of the various components.
 - a Manifest file is included.
- The res folder
 - Contains the not compiled application resources. Resources are the additional files and static content that your code uses, such as bitmaps, layout definitions, user interface strings, animation instructions, and more.

- resource.arsc
 - a file containing pre-compiled resources, such as binary XML for example.
- AndroidManifest.xml
 - An additional Manifest file.
 - Contains name of the package
 - Describes components of the application
 - Required permissions
 - Minimum level of API
- classes.dex
 - This are the classes of the application compiled in the dex file format which is needed for the Dalvik virtual machine.

The most important parts of the APK file that are needed for the research are the AndroidManifest.xml and the classes.dex. When using reverse engineering tools to convert the data to a more readable form [pxb1988,].

2.2.3 The Android Debug Bridge

The Android Debug Bridge [Google, b](ADB) is a command line tool used to communicate with an Android device. This tool can be used to retrieve data from your Android device and is also used to test applications. I will use this to perform certain experiments. It is a client-server program that is installed on a normal desktop computer or laptop. You can then communicate with your android device when it is connected via USB to the computer.

3 The Google Authenticator

The Google Authenticator is an application that implements both the HOTP algorithm and the TOTP algorithm. It is specially made for the two-factor authentication for your Google account, but you are also able to add your own key for One-Time Password generation, so you are able to store all your authenticator keys in one single application that can generate them all. Having a lot of keys on one place means that this application should be well protected because it could be a source of valuable information for people that want to get access to your accounts.

3.1 The installation

The installation of the application is straightforward. You go to the Play store and install the application. At this moment in time the application contains no keys so it is not possible to generate any one time passwords. You have the possibility to store your secret keys via a few different options. In order to use this application for two-factor authentication of your Google account, the option in the option menu of your Google account needs to be activated. Once that has been activated Google can generate a secret key for you.

3.2 Storing your secret key

Before you can use your authenticator you need to have a shared secret between your authenticator and the google server, so you can generate the one time passwords. This should be done in a safe way, and we will evaluate each option made available by Google. The options exist between manual entering or via a QRcode.

3.2.1 Scan a QRcode

Google will you show a barcode which contains all the information necessary for the application to create your account.



Figure 3: QR code generated by Google

When you scan the QR code you will receive this information:

```
otpauth://totp/r.estourgie%40gmail.com?  
secret=eam6n4ej5agecirx&issuer=Google
```

There is no connection needed with the internet. We can see that there is some certain data available. The first part will tell the authenticator that this should be used for TOTP one time password generation:

```
otpauth://totp
```

The second part will show the account name with Augmented BNF encoding [Crocker and P, 2008], in this case r.estourgie@gmail.com:

```
r.estourgie%40gmail.com
```

The next part contains the secret key in base32 encoding [Josefsson, 2003] together with the counter this will be used to generate the HMAC:

```
secret=eam6n4ej5agecirx
```

And the last part shows its issuer, in this case Google:

```
issuer=Google
```

3.2.2 Manually adding a key

The authenticator also accepts keys which are manually inserted. This can be used for other websites which have a 2 way authentication method but don't want to create their own Authenticator application. They are able to provide the user with a key he needs to insert if they don't have the ability to generate QR codes. This is ideal for testing our previous assumption that the secret key of our QR code should be:

```
eam6n4ej5agecirx
```

After the key was inserted we can conclude that this was indeed our secret key, as you can see on the figure below. When you manually add a secret

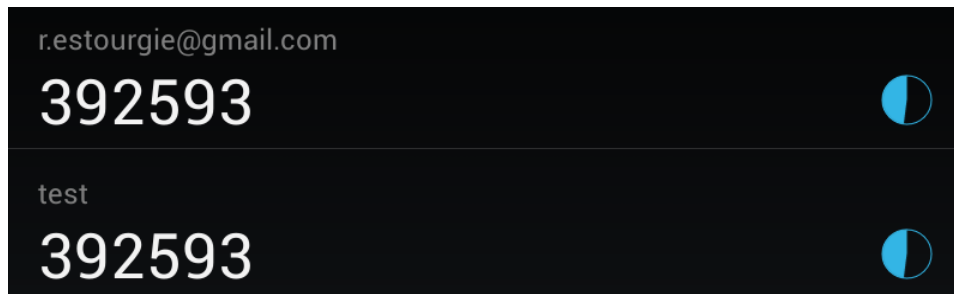


Figure 4: Results scanning QR code and manually entering the code

key you are also able to choose for a HOTP version instead of TOTP.

3.3 Implementation of the protocols

3.3.1 TOTP Protocol

There are a few things that we are able to research here.

- What kind of algorithm do they use to generate the one time passwords?
- How is the time frame implemented?
- Can expired one time passwords be reused?

What kind of algorithm do they use to generate the one time passwords? Google uses key lengths of 16 bytes, according to [MRaihi et al., 2005] 16 bytes is minimum, however 20 is recommended. The documentation states that they always use the SHA1 hashing algorithm for generating the HMAC code. Google uses the android library (javax.crypto.Mac) to create a special MAC object with the secret key. The library has a special HMAC with SHA1 algorithm called HMACSHA1.

How is the time frame implemented, are one time password that are already expired still accepted? The time of one frame is 30 seconds. When trying to log in the maximum frame range is one frame extra. After 1 minute has passed the one time password will not be accepted any more.

3.3.2 Counter based

The HOTP option is available for the authenticator, this is not used for any of Google's services. The algorithm is essentially the same as the TOTP algorithm but instead of a time based increment there is a counter that gets incremented every time the user presses the button on the application.

3.4 Multiple authenticators for one account

Is it possible to have multiple authenticators for one account? This is only possible if you save your QR code that you received the first time you activate your authenticator. This contains your secret key and makes it possible for another authenticator to produce the same One Time Passwords.

3.5 Account Recovery

Google has implemented certain options to recover your account when you forgot your password or lost your authenticator. This is implemented with a security question, recovery e-mail and a personal telephone number. However when an attacker gains access to your Google account, it can have devastating consequences.

3.5.1 consequences

When any of the recovery options is changed the user will receive a notification of this. This notification gives no power to the user to undo the change. If an attacker hijacks your account he can change all these recovery options leaving only one option to the user to retrieve his account. This is done through the Account Recovery Form, it requires you to know a previous password used for your Google account, the last date that you were able to sign in and the date of the creation of your Google account. The first two questions are quite okay for users to answer but the last question is impossible for most of the users to answer. This is time consuming and can lead to a lot of frustration.

3.5.2 A better account recovery solution

The scenario that an attacker can take over an account for a long time needs to be avoided. This can all be solved by a simple authentication solution. Whenever an element of the account recovery options needs to be altered. It needs to be done via the other already available authentication options. This is simple and gives a lot more control to the user.

Changing the old recovery mail When a user changes his recovery mail we can assume that the e-mail account no longer exists or is compromised. When there are no other security features only a password is requested. When there are other security features available one of them is required to validate the change of the e-mail. This comes to filling in the text message received by phone or answering the secret question.

Changing the secret question When a user changes his secret question it's probably not strong enough or more probably the user forgot it. When there are no other security features only a password is requested. When there are other security features available one of them is required to validate the change of the secret question. You can then choose between receiving a text message and filling it in or by receiving a confirmation e-mail with a code.

Changing the mobile telephone number When a user changes his recovery telephone number the worst case scenario is that he doesn't have access to the recovery number any more. When there are no other security features only a password is requested. When there are other security features available one of them is required to validate the change of the telephone number. You can then choose between answering the secret question or filling in the code of the confirmation e-mail.

3.5.3 Conclusion

The current state of the account retrieval security is quite unsafe. It is very easy for an attacker to get complete control over the account by altering all the extra account recovery features. This leaves the user only with the option of the more tedious Account Recovery Form.

3.6 Application-specific passwords

Some specific applications require access to your Google account. This are e-mail clients, youtube clients, Chrome-Synchronise, etc.. Normally they would store your username and password and use it to connect to your Google account. However they do not have any access any more because of the two-factor authentication. To solve this Google provides application-specific passwords. These are very long pass phrases with only intended use for applications. These are generated on the server side of Google. You can generate and revoke them at the authenticator option interface.

When we use this password to log in via the web interface we receive the error shown in the figure below.

This password could be exploited to get full access to your Google account however it doesn't work via the web-interface. However it is not possible via the web interface and this exploit would be nice for future work.

Application-specific passwords

Step 2 of 2: Enter the generated application-specific password

You may now enter your new application-specific password into your application. Note that this password grants complete access to your Google Account. For security reasons, it will not be displayed again:

isfm nhtb vbmo ahkn

No need to memorise this password.
You should only need to enter it once. Spaces don't matter.

Done

| Your application-specific passwords | Creation date | Last used date | |
|-------------------------------------|---------------|----------------|----------------------------|
| test account | 08-Jun-2013 | Unavailable | [Revoke] |

Figure 5: Application-specific password

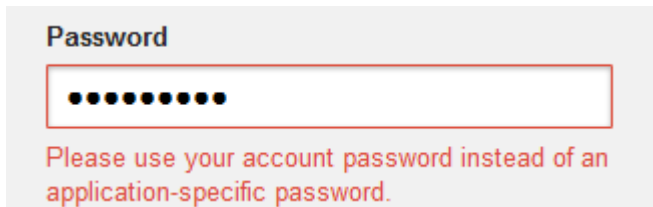


Figure 6: Trying to log in your gmail account with an application-specific password, fails and produces error message

3.7 Trusted Device

Google also made it possible to trust certain devices. This means that you do not need to use your authenticator any more. You go from 2 factor authentication to 1 factor authentication again. This option is given if you log in via the web-interface in the form of a little radio button. After this you don't need to fill in your One-Time Password any more, your password will be enough for this device. The information needed is stored in a cookie,

2-step verification

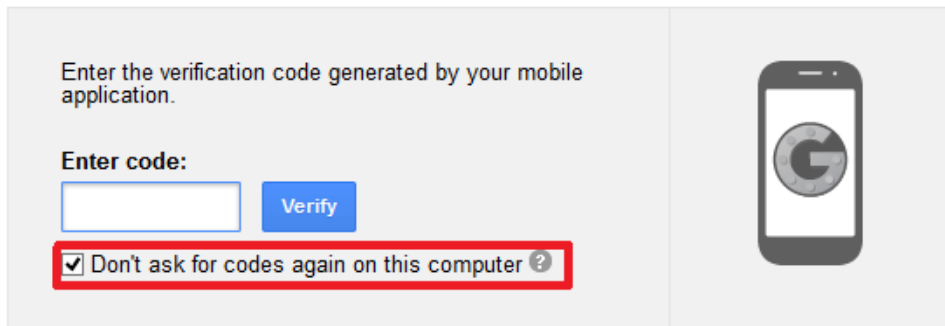
The image shows a screenshot of a Google authentication interface. On the left, there is a text prompt: "Enter the verification code generated by your mobile application." Below this is a label "Enter code:" followed by a text input field and a blue "Verify" button. At the bottom of this section, there is a checkbox with a checkmark and the text "Don't ask for codes again on this computer" followed by a help icon. This checkbox and its text are highlighted with a red rectangular border. On the right side of the interface, there is a graphic of a smartphone displaying the Google logo.

Figure 7: Trusted device radio button

if you delete the cookie you need to re-authenticate yourself.

3.8 Generated One Time Pads on the server

In the account options of Google is an option to generate 10 one time passwords. These one time passwords are meant to be printed out and used in cases of forgetting or losing the Android device. Each code is accepted only once and it doesn't matter in what order. When you decide to generate 10 new codes, the old codes automatically expire.

4 The Battlenet Authenticator

The battlenet Authenticator is an authenticator used by millions of gamers to access their Blizzard games.

4.1 Installation

The installation of the Battlenet authenticator is quite straight forward. You go to the Google play store and install the application. During the installation your key is automatically generated and is linked to a serial number. An example of this is shown in figure 6 4.1. To connect your authenticator to your account you need to visit the Battlenet website, log in to your account and click on add an Authenticator to your account. You will then receive a confirmation e-mail on your linked e-mail account.¹ After you received the e-mail you will be requested to fill in the serial number together with the current one-time password. This is enough for Battlenet to retrieve your secret key.

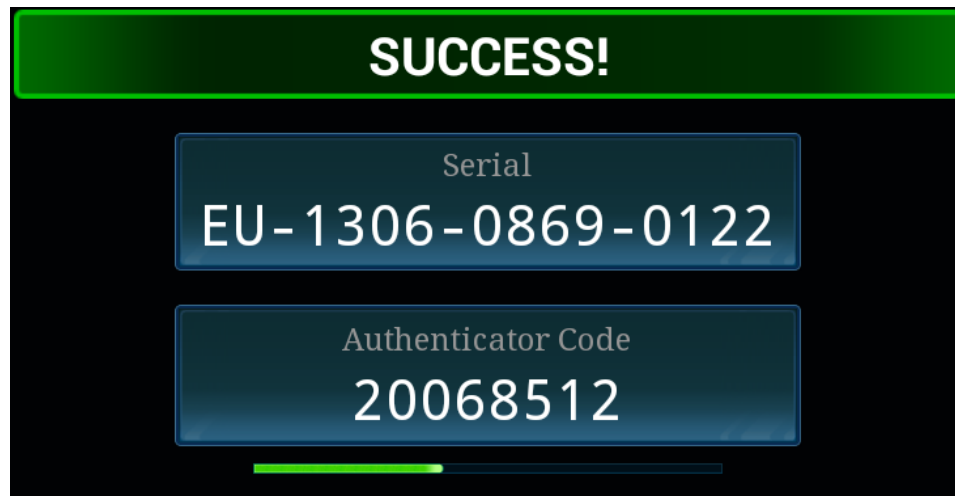


Figure 8: A Battlenet serial number

¹In the past it would occur that an attacker that retrieved your Battlenet account details and would then put an authenticator on your account. This was a frustrating thing to happen because normal account recovery via e-mail wasn't possible because of the authenticator. This could take atleast a day before any Blizzard employee would take a look at your case and remove the authenticator security. Usually your game account would already have been robbed by then. This is from my own experience.

4.2 Storing your secret key

Blizzard has automatically implemented the key storage process. During the first start up of the application the application will require an internet connection. The application will try to make a connection with blizzard and The application will receive a secret key linked to a serial number. The application is now able to generate One Time Passwords based on the key. After the account has been linked to the serial via the web-interface blizzard knows that the secret key linked to the serial is your secret key.

4.3 Implementation of the protocols

The Blizzard Authenticator has no open source code nor has it posted any details about the implementation of the algorithm in the Battlenet Authenticator. It required some reverse engineering to get more specific data about the application. The APK file only contains the classes.dex file. With a special tool (dex2jar) it is converted to a more readable form. The code still contains no comments or explanations, however it is nicely divided in their packages and their classes format.

4.3.1 TOTP protocol

The Battlenet Authenticator implements its own SHA1 algorithm. This is a bit unusual because Android has a supported library to provide SHA1 hashes (java.security.MessageDigest). Even wiser would have been to use the MAC implementation of Android (javax.crypto.Mac) as they did in the Google Authenticator.

How is the time frame implemented, are one time password that are already expired still accepted? The time of the frame is 30 seconds. The battlenet website will accept the One Time Passwords up to 5 frames back. This is 2.50 minute to login.

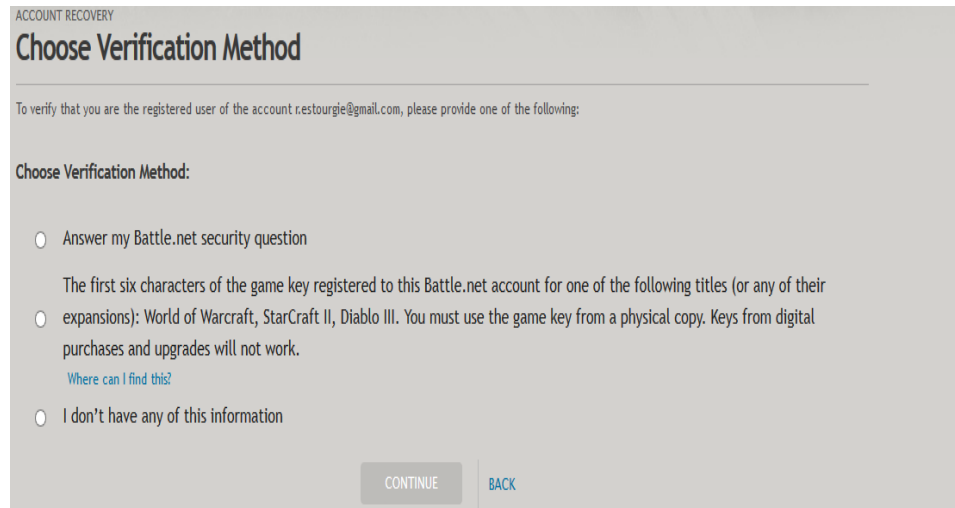
4.4 Multiple authenticators for one account

It is not possible for the Battlenet authenticator to have multiple Authenticators for one account. Every application receives its own key and with it its own serial number.

4.5 Account Recovery

Account recovery is fast and easy on Battlenet. In case of a lost password there are several options.

- You can have a password reset link send to your e-mail address.
- If you lost your authenticator you can have it removed by letting Battlenet send a text message to your telephone. This contains the code you need to fill in to remove the authenticator.
- When you have none of the above you can give them a e-mail address and send in a copy of your identification. A Blizzard employee will then look in to it within 6 to 12 hours. It is impossible to edit your personal information once you have set-up a Battlenet account. So they are always able to identify you via your identification.



ACCOUNT RECOVERY

Choose Verification Method

To verify that you are the registered user of the account `restourgie@gmail.com`, please provide one of the following:

Choose Verification Method:

- Answer my Battle.net security question
- The first six characters of the game key registered to this Battle.net account for one of the following titles (or any of their expansions): World of Warcraft, StarCraft II, Diablo III. You must use the game key from a physical copy. Keys from digital purchases and upgrades will not work.
[Where can I find this?](#)
- I don't have any of this information

CONTINUE | BACK

Figure 9: Battlenet account recovery

4.6 Trusted Device

Just like Google, Battlenet also gives you the possibility to trust a device before you login. This means that you go from 2 factor authentication back to 1 factor authentication. However they have a time frame of thirty days. After thirty days the device will not be trusted any more and is required to re-authenticate the device via the authenticator. The information needed is stored in a cookie, if you delete the cookie you need to use the authenticator again.

5 Security Analysis of the Authenticators

Applications are secured by the operating system, this prevents a lot of unwanted people from accessing your data. In the Android operating system applications get their own user id (UID). This prevents them from accessing each others data. However applications also have a certain role to play in securing their own data. This could be simple things by not storing private keys on public accessible places, or allowing the application to communicate in an insecure way with other applications.

5.1 Android backups

One of the less known new features introduced in Android 4.0 Ice Cream Sandwich (ICS) is the ability to backup a device to a file on your computer via USB [Google, a]. This doesn't require you rooting the Android device and lets you backup application data. The only constraint is to have USB debugging enabled. This enables the adb to communicate to the Android device. When backing up you can include user installed and system installed applications as well as shared storage (SD card) content. There are some limitations to the backup process: Backup can be forbidden when it is explicitly stated in the Manifest of the application. Android wont backup DRM protected applications, system settings such as APNs and WiFi access points. A well designed application with confidential data should have prevented this option by setting the `android:allowBackup="false"` flag in their Manifest [Google, d]. It is also possible to add the entire APK file of the application to your backup. The normal command to backup your application:

```
adb backup app.package.name
```

adb is the Android Debug Bridge. You activate it in the command line tool with the adb commando. It then stores the application (`app.package.name`) in the current directory in a file called `backup.ab`, ab standing for Android Backup. Before it stores the data in the directory it will request you to unlock the device and accept the backup. The Android device gives you the ability to backup the data or to refuse it. It is also possible to encrypt the data, the backup is then encrypted with AES-256 [Daemen and Rijmen, 1998].

The backup.ab file contains the following data:

```
ANDROID BACKUP
1
1
none
"a lot of binary data"
```

The first line defines that it is an Android Backup. The second states the format version, the third is a compression flag, and the last one is the encryption algorithm ('none' or 'AES-256').

The data is visible, as we did not encrypt it. The data is compressed using the Deflate algorithm [Deutsch, 1996]. So the data can be extracted from this file. [Elenkov,] gives an exact description of the extraction. The first four rows contain 24 bytes of data that is not compressed. after skipping the first 24 bytes, openssl zlib converts the Deflated compression to a tar file.

```
dd if=mybackup.ab bs=1 skip=24|openssl zlib -d > mybackup.tar
```

Afterwards you can easily untar it to view its contents. Application data is stored under the app/ directory, starting with a _Manifest file, application files in f/, databases in db/ and shared preferences in sp/. The manifest contains the version code of the application, the platform's version code, a flag indicating whether the archive contains the APK of the application. The end of the _manifest file contains the application signing certificate.

We will now use this technique to test both the Google Authenticator and the Battlenet Authenticator.

5.1.1 The Google Authenticator

The source code of the Google authenticator is openly available to the public, so it is easy to take a look in the Manifest to view that the allowBackup flag is set to false. However this does not stop us from trying to issue the command.

```
D:\Users\Raoul>adb backup -f backups/googlebackup.ab com.google.android.apps.authenticator2
Now unlock your device and confirm the backup operation.
D:\Users\Raoul>
```

Figure 10: Google Authenticator backup

We received a file, apparently we backed something up. Converting the ab file to a tar file also works. However when we try to unpack the file we receive a corrupted file error. After trying to backup a second time the error still persisted. Backing up the same application from another Android device with different data, resulted in the same error. All files are identical and gives enough information to assume that the files contain no special data.

5.1.2 The Battlenet Authenticator

The Battlenet Authenticator application is completely closed source so we need to check if it is backup enabled or not. As we issue the command we atleast receive a package. However this does not mean anything yet as we saw in the previous example of the Google Authenticator.

```
D:\Users\Raoul>adb backup -f backups/blizzardbackup.ab com.blizzard.bma
Now unlock your device and confirm the backup operation.
D:\Users\Raoul>_
```

Figure 11: Battlenet Authenticator backup

Extracting of the data of the tar file out of the ab file is successful. Extracting of the tar file also works. The Battlenet Authenticator backed up the manifest files and two files in the shared preference folder. The definition of the sharedpreference [Google, e] on the Android developer site is:

The SharedPreferences class provides a general framework that allows you to save and retrieve persistent key-value pairs of primitive data types. You can use SharedPreferences to save any primitive data: booleans, floats, ints, longs, and strings. This data will persist across user sessions (even if your application is killed).

This might be a place to find any stored keys. By backing up a second blizzard application with a different key we can view the differences with WinMerge. The file SplashActivity of the second backup is identical to the SplashActivity file of the other backup. The com.blizzard.bma.AUTH_STORE file is totally different. The backup file can be found in the appendix 9.1

5.1.3 Restoring a backup file

It is really easy to restore a backup file. It does not require any of the unpacking steps. You just need an Android device and the Android backup file. When accessing the adb you fill in:

```
adb restore mybackup.ab
```

The Android device will then give a notification. The application will now be in exactly the same state as the backed up Android application. This means that you now have an identical authenticator generating the same codes for the Battlenet Authenticator.

5.2 Debuggable android applications

This is another option that might work on an Android device with USB debugging enabled. This option is the run-as command, this command only works when the application is debuggable. The run-as command tries to run itself as the application itself, this also means that it has access to all the data. We then can copy the data to a place where we are able to access it. This command is enabled by setting the android:debuggable flag to true, it is only intended to be used when you are still in the development phase of the application. The default option is false so it should be stated explicitly in the Android Manifest. When the application is released on the market the android:debuggable="true" should be removed.

5.2.1 The Google Authenticator

The Google Authenticator prevented this option. So we are unable to run as the program itself.

```
shell@android:/ $ run-as com.google.android.apps.authenticator2
run-as com.google.android.apps.authenticator2
run-as: Package 'com.google.android.apps.authenticator2' is not debuggable
!shell@android:/ $ _
```

5.2.2 The Battlenet Authenticator

The run-as command seems to work for the battlenet authenticator. This means that we can run as the application itself and are able to access the key. Now we can use the cat command to copy the key to a public accessible location like the SDcard. This then gives us the option to retrieve the data from the SDcard.

```
D:\Users\Raoul>adb shell
shell@android:/ $ run-as com.blizzard.bma
run-as com.blizzard.bma
shell@android:/data/data/com.blizzard.bma/shared_prefs $ cat com.blizzard.bma.AUTH_STORE.xml > /sdcard/bliz.xml
zzard.bma.AUTH_STORE.xml > /sdcard/bliz.xml <
shell@android:/data/data/com.blizzard.bma/shared_prefs $ exit
D:\Users\Raoul>adb pull /sdcard/bliz.xml
0 KB/s (459 bytes in 1.000s)
```

5.3 Key Storage

Even-though an application might seem secure, we cannot guarantee that the operating system is secure. It is in the best interest of the application and the user to make it as hard as possible for an intruder to retrieve confidential data. In this section we assume that the attacker has full root privilege on the Android device. We can then see how hard it is to retrieve the actual key that is stored by the application.

5.3.1 The Google Authenticator

The Google Authenticator is secure if the operating system is secure. However the data in the database of the Google Authenticator is in plaintext. It takes no effort to link the key to the right username.

5.3.2 The Battlenet Authenticator

The Battlenet Authenticator is not that secure, however the key data in the xml file is obfuscated a bit. An attacker would not be able to see it in an instance. However if the application would be reverse engineered and the attacker would put some time in it beforehand the key would also be extracted fast.

6 Improving Security

The hardest part in the security is not performing the cryptographic operations, but key management. If a key is stored along with the encrypted data, it is fairly easy to extract it, especially on a rooted device. The same is true for keys embedded in the application source code, even if they are somewhat obfuscated. The applications could make better use of the secure alternatives offered by Android. These implementations would make both the Google Authenticator and the Battlenet Authenticator more secure.

6.1 Pin access

To improve security the application could require a pass-phrase or PIN code to access the application. This would be a good security even in rooted devices. Because it would require a code not available in the device itself.

6.2 Android Keychain API

Since Android 4.0, Android offers a KeyChain [Google, c]. Currently the Android Keychain API can only be used to store RSA private keys and certificates. It is not yet generic enough to allow secure storage of user data like symmetric keys. However chances are there that this option will be available in the future so a small overview of the Keychain API will be given.

When the KeyChain API is started for the first time, it will ask for an user password. The KeyChain has as main advantage that the user only needs to remember one password, all other credentials can then be stored in the KeyChain.

When an application stores its key via the KeyChain API it will be encrypted with a 128-bit AES master key in CBC mode [Morris, 2010]. The data then contains an info header, the Initial Vector(used for encryption), an MD5 hash value of the data and the encrypted data itself. The master key is itself encrypted with an AES key that is derived from the password filled in by the user. To derive a key from the password PBKDF2 [Kaliskir, 2000] with 8192 iterations is used together with a random salt. PBKDF2 is used to stretch the password to the correct size and the random salt is used to prevent any lookup table attack.

So in fact this is not so different from the PIN access mode, this is however a system-wide solution.

6.2.1 The Secure Element

The Secure element is a tamper resistant smart card chip. It is a small computing environment on a single chip. Recent cards also come equipped with cryptographic co-processors implementing algorithms like DES, AES and RSA. They take advantage of the hardware's memory protection features to ensure that each application's data is only available to itself. This would be an ideal solution for the authenticators. If you generate the One Time Password on the Secure Element it would make the application a lot more secure. Leaving the application itself unable to request the key from the Secure Element it would be impossible for an attacker to get a hold on to the key. The only possibility would be to request a temporary One Time Password, granting access only for a short period of time.

7 Conclusion

7.1 The Google Authenticator

The Google Authenticator is a well designed application, because it is open-source anyone can view the code and report bugs if there are any. However the key storage is a big security risk. Using one of the alternatives in the security improvement sector would be strongly recommended.

7.2 The Battlenet Authenticator

The Battlenet Authenticator should have some major improvements. They have forgotten about some serious issues in the security of the application. The only good thing they have done is to obfuscate the secret key a bit. That is however only a temporary solution and wont hold back a good hacker that long. They can better fix the security holes and implement an improved security solution in the Improved Security section.

7.3 Trusted Device

The trusted device option is insecure, a man in the middle attack can be used to get access to the account. This could be in an internet caf where the owner redirects the user to a modified version of the website, the user would not notice it and would just continue logging in. This kind of attacks cannot be prevented with a One Time Password. However because of the trusted device option the attacker can have access 30 days for (Battlenet) or an undefined time (Google).

The application should be convenient to use, so it is understandable that they have implemented it. There is a safe solution to prevent this attack. This can be done by requesting another One Time Password to confirm that this device is really trusted.

8 Future Work

There is still a lot of research that can be done on the security analysis of both applications. A further evaluation of the Battlenet application is required to see how they acquire the secret key, and the exact way of storing the key is also important.

It is also interesting to do research on encrypted backup's, how are they exactly encrypted and are they easily decrypted?

Trusted devices are identified by cookies, is there an exploit that you can use the cookie of a trusted device, for a not trusted device?

The application specific password from Google, grants access to a Google account. Using this password via the web interface does not work. How is this implemented?

9 Appendix

9.1 Battlenet backup file

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<long name="com.blizzard.bma.AUTHSTORE.CLOCK_OFFSET" value="3536" />
<int name="com.blizzard.bma.AUTHSTORE.HASH_VERSION" value="10" />
<string name="com.blizzard.bma.AUTHSTORE.HASH">08ed449963465c5
35703d3d513c0f00966f42842105cea428896ae98e00a38e7fd384a8bb50a15
0ee0ce894d24fa5a4949396a2476bbbe5baa</string>
<long name="com.blizzard.bma.AUTHSTORE.LAST_MODIFIED" value="1
370728049275" />
</map>
```

References

[Alugbue,] Alugbue, R. Apple is the top smartphone vendor for 2011, but android os continues to lead market share. <http://tinyurl.com/a375fdq>.

[bbc,] bbc. 900 million activations and 48 billion installed applications. <http://www.bbc.co.uk/news/technology-22542725>.

[Bellare et al., 1996] Bellare, M., Canetti, R., and Krawczyk, H. (1996). Keying hash functions for message authentication. In *Advances in Cryptology CRYPTO96*, pages 1–15. Springer.

- [Crocker and P, 2008] Crocker, D. and P, O. (2008). RFC4234 augmented bnf for syntax specifications: ABNF. In *Internet Request for Comments*.
- [Daemen and Rijmen, 1998] Daemen, J. and Rijmen, V. (1998). Aes proposal: Rijndael. In *First Advanced Encryption Standard (AES) Conference*.
- [Deutsch, 1996] Deutsch, L. P. (1996). DEFLATE compressed data format specification version 1.3.
- [Elenkov,] Elenkov, N. Unpacking android backups. <http://tinyurl.com/mdd8ahd>.
- [Google, a] Google. Android data backup. <http://tinyurl.com/28dqrpf>.
- [Google, b] Google. The android debug bridge. <http://tinyurl.com/bnxdazm>.
- [Google, c] Google. Android keychain. <http://tinyurl.com/6lkw19v>.
- [Google, d] Google. Android manifest. <http://tinyurl.com/1oxzpnh>.
- [Google, e] Google. sharedpreferences. <http://tinyurl.com/37qe4s4>.
- [Josefsson, 2003] Josefsson, S. (2003). RFC3548 the base16, base32, and base64 data encodings. In *Internet Request for Comments*.
- [Kaliskir, 2000] Kaliskir, B. (2000). RFC2898 PKCS 5: Password-Based Cryptography Specification Version 2.0. In *Internet Request for Comments*.
- [Krawczyk et al., 1997] Krawczyk, H., Bellare, M., and Canetti, R. (1997). RFC2104 hmac: Keyed-hashing for message authentication. In *Internet Request for Comments*.
- [Morris, 2010] Morris, D. (2010). Recommendation for Block Cipher Modes of Operation: The XTS-AES mode for Confidentiality on Storage Devices.
- [MRaihi et al., 2005] MRaihi, D., Machani, S., Pei, M., and Rydell, J. (2005). RFC4226an Hmac-based One-Time Password algorithm(HOTP). In *Internet Request for Comments*.
- [MRaihi et al., 2010] MRaihi, D., Machani, S., Pei, M., and Rydell, J. (2010). RFC6238Time-based One-Time Password algorithm(TOTP). In *Internet Request for Comments*.

[pxb1988,] pxb1988. Dex2jar. <http://code.google.com/p/dex2jar/>.

[Shabtai et al., 2010] Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., Dolev, S., and Glezer, C. (2010). Google Android: A comprehensive security assessment. *Security & Privacy, IEEE*, 8(2):35–44.