Permissionless Banking API

Christos Kyprianou

Delft University of Technology

Permissionless Banking API

Master's Thesis in Embedded Systems

Parallel and Distributed Systems group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Christos Kyprianou

Jan 31, 2018

Author
Christos Kyprianou

Title
Permisionless Banking API

MSc presentation
Jan 31, 2018

Graduation Committee

dr. ir. J. A. Pouwelse	Delft University of Technology
dr. J.S. Rellermeyer	Delft University of Technology
dr. Z. Erkin	Delft University of Technology

Abstract

In this document we present an in-depth vulnerability assessment of the HSBC banking system. Going beyond prior work, we investigate the full stack down to the Java bytecode level and further analyze its 3 key platforms: Android OS, IOS and online banking. During the process we analyze their main security feature, OTP generation algorithm and examine the inner workings of mobile/online banking through various vulnerability assesment techniques and successful man in the browser attacks in their main platforms showing the client-server communication packet flow. During the process we discovered several vulnerabilities and show that HSBC leaks details to multiple webserver which are not under it's direct control such as: lo.v.liveperson.net. Additionally, the HSBC app is found to be inefficient, for instance, it repeatedly sends "Lorum Ipsum" phrases. A gross waste of bandwidth, bordering on incompetence. Finally we present our own version that is able to perform the basic banking functions: 1-log-in, 2-view accounts and balances, 3-view transaction history and 4-perform transactions, adding more than 200% speed improvement.

Preface

This is the report of my MSc thesis project on implementing and testing a permissionless banking API, for the completion of my MSc degree in Embedded Systems. This research has been carried out with the Parallel and Distributed Systems research group of the Faculty of Electrical Engineering, Mathematics, and Computer Science of Delft University of Technology, supervised by J. A. Pouwelse.

I would like to thank my supervisor, Johan Pouwelse, for giving me the opportunity to work on such exciting project and the motivation he inspired during our meetings. Further, I would like to thank Egbert Bouman and Martijn de Vos for sharing their knowledge and experience, helping me make this work possible.

Christos Kyprianou

Delft, The Netherlands
26th January 2018

Responsible Disclosure

This work is part of a greater project from the research group of Parallel and Distributed Systems of the Faculty of Electrical Engineering, Mathematics, and Computer Science of Delft University of Technology which studies the performance and feasibility of an open banking environment. During this work we point out vulnerabilities and potential critical security issues found in the inspected app, and this report can be considered as a complete summary to help solving the potential security issues.

To protect the code safety, we DID NOT work with the original Java or C/C++ source code, but only decompiled code from the Android APK files which are publicly available. In this report we only provide high level descriptions of algorithms and vulnerabilities, without giving functioning code parts or detailed descriptions, keeping this way important algorithms and code confidential.

Contents

Preface **v**

Responsible Disclosure **vii**

1 Introduction **1**

2 Research Goals **3**
- 2.1 Faults of the past 3
- 2.2 History repeats 4
- 2.3 European pressure for change 4

3 Vulnerability assessment tools **7**
- 3.1 Android debug bridge - ADB 7
- 3.2 Androguard 7
- 3.3 Smali/BackSmali 8
- 3.4 Deobfuscation tools 8
- 3.5 ApkTool 8
- 3.6 Dex2jar and JD-GUI 8
- 3.7 APK debugging 9
  - 3.7.1 JDB-Java debbuger 9
  - 3.7.2 IDA PRO debugging 10
- 3.8 Drozer 10
- 3.9 Data storage/transfer vulnerabilities 10
  - 3.9.1 Side Channel Data Leakage 10
  - 3.9.2 Shared preferences 11
  - 3.9.3 SQLite and NoSQL databases 11
  - 3.9.4 Internal and External storage 11
  - 3.9.5 User Dictionary cache 11
  - 3.9.6 Android adb backup 11
- 3.10 Encryption 12

4 Authentication using tokens **13**
- 4.1 Authentication practices 13
- 4.2 OTP generator 14

4.2.1 Get token Key 17
4.2.2 Generate time-stamp 18
4.2.3 Generate hash code 18
4.2.4 Generate code 18
4.2.5 Truncate code 18

5 HSBC vulnerability assessment 19
5.1 HSBC mobile banking 19
5.2 Starting point - Transferring registration 20
5.3 Exploring Android app 23
5.4 Man-in-the-Middle in iOS 25
5.5 Man-in-the-Middle in full-access online banking 25

6 HSBC mobile & online banking 29
6.1 Mobile banking walk-through 29
6.2 Online banking walk-through 32
6.3 Security assessment 35

7 Vulnerability assessment of HSBC OTP generation 41
7.1 Yet another APK update 41
7.2 Locating OTP generator 42
7.3 OTP function calls 43
7.4 HSBC code protection mechanisms 45
7.4.1 Renaming Identifiers 45
7.4.2 Call Indirection 46
7.4.3 Code reordering 46
7.4.4 Arithmetic obfuscation 46
7.4.5 Really complex call graph 48
7.5 OTP main algorithm 48

8 HSBC open-source banking 51
8.1 Application and code size 52
8.2 Function calls 52
8.2.1 HSBC_manager 52
8.2.2 HSBC_requests 53

9 Permisionless banking API evaluation 55
9.1 Un-necessary data 55
9.2 Performance evaluation 60
9.3 Flexibility 60

10 Conclusion 63

Chapter 1

Introduction

Every day we entrust our life earnings to banks and manage our finances through bank accounts. Banks put an effort into making money handling easier for their customers by giving access to accounts through mobile and web banking applications while employing mechanisms to keep the accounts secure from malicious persons. The collapse of Cyprus banking system in 2013, has shown to the world that banks can not be completely trusted for keeping public's money safe [1]. Unlike this one occasion, the security risks from malicious attacks or exploits from poor app design make up, an even worst danger to be concerned.

In previous work of our research group [2], the today's dangers in mobile banking applications are presented, coming from attacks such as Man-in-the-middle (MITM) [11], rogue access point[10] and the mallware targeting both, mobile banking applications [4, 5] and Android in general [6, 3]. Following to that, in "Vulnerability Analysis of Banking Applications on Android Smartphones", [2] a vulnerability assessment on the Android banking application of Bank of Cyprus takes place. Similar work is done by J. Awsome et al. and J. Doe et al. [8, 9] for Mobile banking applications of ING, Rabobank, ABN AMRO and German open HBCI/FinTS protocol. This work can be considered as a follow-up on vulnerability analysis on mobile banking applications with UK's HSBC bank as a target [17]. However, this time instead of singling out a platform for the analysis, we explore all options including the banking applications on Android, iOS and website since a system is only as secure as its weakest link. Our findings show that HSBC has put considerable effort to secure the Android system while iOS app security can easily be bypassed with a jail-broken phone and a certificate killing app. On the other hand website is proved to be the most vulnerable since there is no protection against man in the browser attacks. Furthermore, as our project capstone, we introduce our own open version of the HSBC banking application which is able to perform the basic functions: *1-Log-in, 2-View Balance, 3- View transactions and 4- make transactions*, and limits the redundant information/packets during client-server communication.

The rest of the thesis is structured as follows: In section 2 we discuss the goals and expectations of our research. In section 3, we present a collection of Android hacking tools and vulnerabilities that are easily accessible and freely available in the internet, require no root access for the attacker, and are simple enough even for a beginner to use. Moving to section 4, we discuss the inner workings of one time passwords (OTPs) [18] that are used by most of the banks today either in special devices [19] or in mobile applications. Furthermore, our vulnerability analysis is presented in sections 5, 6, and 7, where we go through the security of mobile/online banking platforms, present their inner workings and OTP algorithm. In section 8, we introduce our permissionless banking API version of HSBC and then compare it with the current banking options in secton 9. Finally, the key findings and conclusions are discussed in section 10.

Chapter 2

Research Goals

In this section we will discuss how banks handled failures and money loss in the last three decades, and their behaviour towards the customers. Further, we propose a method which could result in faster failure identificaton and improvment, thus better customer protection and satisfaction.

2.1 Faults of the past

In a paper published in 1993, R. Anderson discusses about why cryptosystems and, more specific, banking cryptosystems fail [12]. According to Anderson, unlike other engineering problems, state of the art cryptosystems keep failure reports secret, while only outdated systems might publish the causes of a crash. Furthermore, bank behaviour in cases of frauds differs from country to country. On the one hand, US regulation demands banks to refund customers for all disputed transactions unless they can prove the customer has committed a fraud, which pushed the development of advanced security mechanisms. On the other hand, Britain bankers, due to lack of strict customer protection regulations, deny faults from their systems, resulting into people being charged for crimes they did not commit.

R. Anderson provides a plethora of fraud examples on ATM machines conducted by working employees, maintenance engineers even outsiders. In some cases customers lost large amounts of money while bank kept blaming them for fraud. Further research shows that techniques used almost three decades ago still exist today with a prime example ”jackpotting”. Jackpotting is a technique where the attacker can replay the communication sessions between ATM and bank in order to withdraw money from the machine multiple times [12]. Similar attacks still exist today through many malware families, including ALICE which is discovered recently and stands out for its simplicity [13].

Besides attacks on ATM, R. Anderson discusses how poor decisions in PIN generation and management increase the chances of the code being guessed, decrypted, and even result into exposing the account number. With this, Anderson notes the importance of having security experts review how these procedures should be performed.

As a consequence of those failures along with bank behaviour, the author explains that banks put customers in a position that they cannot prove that a failure occurred, while it creates an encouraging factor for bank employees to exploit their position and commit fraud. Furthermore, the lack of keeping central records from customer complaints makes frauds possible to be repeated without notice.

Finally, the author proposes two approaches to improve systems. First, by formal verification of hardware and software safety features, and second, through constant feedback and improvement.

2.2 History repeats

In a more recent time-line, technological progress like online/mobile banking makes banking fraud and money theft even easier, since the attacks can be performed from remote location and target larger groups of people. According to Brignall [14], online bank fraud is the "UK's fastest growing area of crime", with £25m being lost due to scams among 5000 customers in the first 9 months of 2015. While the bank refuses any refunds for around 70% of those customers, security experts claim that, most of the scams could be prevented if the banks matched the payee account with account name. However, instead of taking action, government rules demand that fraud cases should be reported to banks instead of police. This results to banks labeling reported frauds as customers fault, lowering this way the crime figures. Even in the current year, UK banks still keep the same mentality and claim that in cases of fraud, victims "should accept the blame and not expect automatic refunds" [15].

Considering that bank fraud is still an ongoing problem, while banks have turned their backs against customers-victims we review R. Anderson approaches for improving the systems in view of recent technological development. Starting with formal verification is something that bank employees should do before releasing their systems thus there is nothing outsiders can do about it. However, Anderson's second approach, *feedback and improvement* is something that gets a new meaning in today's society. Employing open protocols and open standards, is something that could benefit both bank and customers. This way, outsiders can view and test the banking APIs and report any vulnerabilities or bugs they find before an exploit is implemented. Furthermore, the customers can know exactly what happens behind the interface and machine they use to manage their account.

2.3 European pressure for change

This approach comes a bit closer to reality with European union's *Payment Services Directive (PSD2)* law that is applied since January 2016 and EU countries have to incorporate it into national law by 13 January 2018 [16]. PSD2 law aims to force banks into allowing a secure way for customers to authorize a third party provider to provide access to account, and transaction data, or make payments, without other parties being involved. This puts customers in control, since it creates more options for managing their finances, bank's banking platform or third party applications. By this law, banks across EU have to provide a public API that can perform the noted tasks for third parties to use. This could impact the way payments are made since a payment can be done directly from the bank without any intermediate companies involved. Furthermore, account aggregation becomes possible since an application could use APIs from multiple banks, having this way all the accounts stored in one application.

However, the law alone cannot fix everything since bank behaviour will play a significant role. Banks could provide the minimum requirements and then keep a defensive stance against third parties while keep their old ways of working, or embrace the opportunity, accept feedback, and improve their business model and systems, in cooperation with the third parties.

Having covered those topics, with the deadline of PSD2 law incorporation this January, our research lies on how open APIs and constant feedback can improve the current banking system. For that reason, we chose a well known, globally operating bank, HSBC, in order to analyze their mobile and online API along with their key generating algorithms and develop our own open API in an attempt to evaluate whether there is space for improvement in security, usability and performance.

Chapter 3

Vulnerability assessment tools

While examining the HSBC mobile banking app on Android, all operations were done on a non-rooted device. Even though rooting a device provides more efficient methods/tools when analyzing an Android app and a significant increase in rooting malware is noticed in the last years [20], not every device in the field is rooted. In order to include all Android devices in our research we decided to conduct our vulnerability analysis on the HSBC mobile banking app on Android on an unmodified Nexus 6 device.

Interestingly, after conducting a research on vulnerability assessment tools, even for non-rooted devices, we came across countless tutorials for numerous applications designed to be used even by inexperienced users. Due to this plethora of tools and vulnerabilities, this section is dedicated to present some worth mentioning findings of our search.

3.1 Android debug bridge - ADB

Android debug bridge (ADB) is a command-line tool that performs communication between Android device and user [21]. Using adb, one can install or remove applications from a device, create backups, and restore them, even navigate through the device folders, and send or receive files from device to computer and vice versa.

3.2 Androguard

Androguard [22] is a python based tool set to be used for static analysis [23]. It can be installed individually and is also included in the virtual machines A.R.E. (Android Reverse Engineering) [24] Santoku [25] running on linux. Androguard supports three compilers that can be used for .APK, .DVM, .dex, .class files resulting to a more readable .Java code. Other functions of Androguard are included in table 3.1

Tool	Purpose
Androlyze	1) Decompile: DAD, dex2jar+jadx, DED option, 2) Get list of: classes, permissions, strings, activities, methods.
Androaxml	View AndroidManifest.xml
Androsim	Compare two apk files
Apkviewer	View methood calls in mathematical graphs
Androapkinfo	Information about apk file

Table 3.1: Androguard tools.

3.3 Smali/BackSmali

Smali [26] is an assembler used for the .dex format used by Dalvik [35] Java Virtual machine that is used by Android and BackSmali is its disassembler [27]. Even though Smali files are difficult to read, understanding the code is possible. For a better understanding of Smali code, lohan provides an example of Smali code in use in his blog [28].

3.4 Deobfuscation tools

Simplify [29] is a generic Android deobfuscator. It virtually executes the selected app and then, based on its behaviour, optimizes the code to be human readable while maintaining the original semantics. Like simplify, JMD [30] is also a Java bytecode deobfuscation tool for general purposes.

3.5 ApkTool

ApkTool [31] is an essential tool for analyzing Android binary apps. It is used for disassembling/rebuilding resources from/to APK binaries. After disassembling an APK file, the user can then decompile using Dex2jar and JD-GUI in order to read the code in Java language. Also user can add patches to the disassembled bytecode. In the case where the user added a patch to the code they can rebuild the APK but before installing it back to the mobile device they have to sign the APK [32].

3.6 Dex2jar and JD-GUI

Dex2jar [33] and JD-GUI [34] are decompiling tools. The first is used to decompile the Dalvik executable, classes.dex, to Java bytecode in .jar format. The latter is a lightweight GUI to help navigating through the Java code.

3.7 APK debugging

An Android vulnerability assesment tool discussion would not be complete without included application debugging. On Android, is possible to perform debugging during run-time if the apk is set to debuggable. This Android feature helps catching bugs but also provides a great way of understanding the inner workings of an application and spotting vulnerabilities. Android application debugging can be achieved using multiple methods for byte code up to Java code debugging.

3.7.1 JDB-Java debbuger

JDB [36] is a command-line debbuger for Java classes that comes preinstalled with JDK. Using JDB, the user is also able to inject code during the run time of the applicaton by altering variable values. Basic functions/commands of JDB are the following:

- run - start JVM
- stop in <function> - Set breakpoint
- list - Show source code around current location
- clear - Show all breakpoints set
- threadgroups - Show threadgroups
- threads - Show threads
- classes - Show all classes loaded in JVM
- class $< classname >$ - Show info about class
- methods $< classname >$ - Show methods of class
- next - Execute next line
- step - Get into method
- step up - Get out of method
- where - Command shows the current call stack.
- locals - See local variables (if compiled with -g)
- stop in - Set breakpoint on method
- print $< variavle >$ - Print variable value
- set $< variable >$ = ... - Change variable value

3.7.2 IDA PRO debugging

Ida pro [37] is an advance and well known tool for disassembling and debugging applications. Use of this tool makes it possible to debug Android byte-code at run-time [38].

3.8 Drozer

Drozer is an open source software attack framework for Android that allows the user to interact with other Android apps [39]. It can be used on both emulators and Android devices to find, test and exploit Android vulnerabilities. As an example, one of the most common uses of Drozer is exploiting Android's Inter-Process Communication in order to extract sensitive information that an app installed on the device makes accessible to other apps [40].

3.9 Data storage/transfer vulnerabilities

Besides the applications and attack tools, the whole concept of a portable device such as a mobile phone itself can become a significant threat when it comes to sensitive data and storage. As mentioned earlier, Drozer is one app that can exploit the communication between Android process to capture sensitive information that an app shared. However, this is just one of the many points that an attacker could start when examining an app. Infosec discusses about side channel data leakage and their Androud hacking & security tutorial series [41, 42] . Furthermore, in their 2016 report, OWASP Mobile Security Project - Top Ten Mobile Risks [43], has rated insecure data storage and unintended data leakage as the second highest risk below improper use of platform. Thus, unintended data leakage and easily accessible data storage are a crucial issues of data security and should be treated accordingly when designing an Android app.

3.9.1 Side Channel Data Leakage

Data leakage through side channels refers to inputs/outputs that a process might send or receive and that may be easily accessed to third parties. This sources might be copy/paste data, application logs, even data analytic or browser cookies. As an example to the latter, in their article - vulnerability analysis of mobile banking applications [2], the author show that after a successful log-in, the authentication of the server to the client is depending on a single cookie. As a result, any person that gets to know the cookie can have access to the mobile banking without ever knowing the credentials of the user.

3.9.2 Shared preferences

SharedPreferences falls into the category of data local storage. SharedPrefereces files are XML files containing values that are used by the app. They are stored in the path /data/data/<package name>/shared_prefs/<filename.xml> and according to the *Android developers getting started tutorial*, the SharedPreferences API should be used when *saving a small collection of Key-Value Sets* [44]. However, knowing the path of the files, a simple adb command allows an attacker with physical acces to the device to extract, read and modify the files as shown by J Rizzo [45].

In order to protect SharedPreferences, the Secure-preferences library [46] , provides an encryption wrapper that encrypts sharedPreferences values stored in .XML files. Even though this will increase the security of sharedPreferences, the encryption keys are still stored on the device and anybody with root access could extract them.

3.9.3 SQLite and NoSQL databases

Similar to SharedPreferences, SQLite and NoSQL are used for storing data into databases [47], [50] . Those databases are located in the path /data/data/<package name>/databases
/<databasename.db> and can also be extracted using adb.

3.9.4 Internal and External storage

Internal and external storage refers to device storage space or SD card. After a simple search in the Smali code using keywords such as ”toString()”, *.txt, ”open-FileOutput”, ”FileOutputStream” or ”sdcard” can reveal whether information is stored in those storage spaces along with the path to reach them. When storing data on SD card one should always keep in mind that the card can be removed from the device and tampered with by an adversary with physical access to the device. Furthermore, Srinivas [49] demonstrates an attack where a simple patch on an app can change storage path to SD card instead of internal storage.

3.9.5 User Dictionary cache

According to Srinivas [49], Android feature, user dictionary, where user stores frequently used or new words, can also store sensitive information that the user typed in. This information is stored in a single database and can be accessed by any Android application.

3.9.6 Android adb backup

Android backup [51], is a way to extract an Android application including its stored data using Android Debug Bridge (ADB). By extracting the app and data on a computer, an attacker can read the data and APK code, or alter them to replace the old app with a patched version.

3.10 Encryption

Encryption libraries can be used to protect localy stored data [46, 48]. Encryption methoods can increase the security of an app only when they are used in a correct way. The two most common mistakes when using cryptography according to Srinivas [52] are:

- Weak encryption algorithms or the use of encoding as encryption that can be easily be decrypted/decoded.
- Weak implementation on strong encryption algorithms - i.e. hard code the keys into the app.

Chapter 4

Authentication using tokens

4.1 Authentication practices

Over the years banks have introduced multiple mechanisms for authenticating the user when online banking is used [53]. As part of the bank authentication evolution, we have seen PIN codes and password phrases, memorable questions, use of transaction authentication codes (TAN) or one time passwords (OTP) and two factor authentication techniques. The use of PIN code, an information that only the authorized client should know, in combination of a TAN code generated by a physical device that only the client is supposed to have, falls into the two factor authentication (TFA) category [54]. TFA is considered the current state of the art in authentication used by not only banks but tech companies as well including Google, Facebook and Apple [57].

In general, in authentication systems, PIN and TAN codes are combined in different ways in order to enhance the security of the generated codes under MITM and phishing attacks. In [56], RedTeam Pentesting GmbH, discuss the vulnerabilities PIN/TAN, PIN/iTAN and PIN/chipTAN authentication systems under MITM thread.

Starting with PIN/TAN [55], a currently outdated method, the client obtains a list of TNAs that each be used only once. In the case of PIN/TAN, if any of the TAN codes becomes known along with the client's credentials, it can be used by a malicious person to perform unauthorized transactions. J. Buchmann et al. in [58], shows that this system is vulnerable to MITM and phishing attacks by attacking banking applications that support PIN/TAN authentication.

As an extra step of security, indexed TAN (iTAN) [55] is used to protect the client of phishing attacks that may obtain TAN codes. In PIN/iTAN systems, the TAN codes are indexed and bank requests a specific TAN code for a transaction. This way intercepted TAN codes cannot work for other transactions. However, this system is still vulnerable to MITM attacks where the attacker has control over client-bank communication and can make the user believe he gives the iTAN for his transactions but providing the iTAN for an unauthorized transaction instead [56].

Moving to today's technologies, TAN or OTP codes can be sent to a registered mobile device that the user owns through a "secure" channel in the form of SMS. This is also known as mTAN method [55]. According to C. Serrato [59] this technique includes three main drawbacks. First the system may put extra charges to clients for the SMS and the availability of the technology depends on the area network coverage. Finally, this system is still vulnerable to MITM attacks that target mobile devices message services. An example of such attack is described in [60], where Zeus botnet was used to steal $47M from European bank customers.

An attempt to eliminate the MITM thread from capturing or misusing the TAN codes, a trusted device (hardware token) that is responsible to generate the desired codes [55] is given from the bank to the customer. This devices come in different variations based on the bank's security standards. Some of them can calculate a sequence and provide an OTP while others are able to respond to authentication challenges based on a common secret [61]. ChipTAN is an example of such application where the user receives a calculator-like device from the bank [56]. ChipTAN codes are generated with the user inserting his bank card inside the device and also providing information regarding the specific transaction. This way for a MITM attack to succeed, should be targeting both transaction data and device data. However, according to [56], chipTAN system is still vulnerable to multiple types of transactions since an attacker could still change some of the transaction data that are not included in the TAN code generation.

Even though use of a hardware token proves to increase security it still has a functional drawback, the user needs to have the physical device with him every time there is a need for an OTP generation. For this reason there are banks including HSBC, that implement their OTP generator within a mobile app using software (soft token). The advantages of this method is its simplicity since no extra hardware is needed besides a smart phone. In addition to that, not having to buy extra equipment or be charged for SMS codes, soft token minimizes the costs of an OTP generator and is also immune to network coverage since the OTP is generated locally through software [59].

A soft token authentication mechanism could perform in the same manner of a hardware token but when it comes to security, since the codes generators operate in mobile platforms, including Android, all vulnerabilities and threads discussed in section 3 along with more sophisticated attacks and malware may used against it. For that reason a poorly designed soft token mechanism instead of securing the client could leave him vulnerable in the hands of attackers

4.2 OTP generator

In the previous paragraphs we discussed the advances in threads and security that led to OTP and their evolution. Considering hard and soft tokens being the preferred OTP generation solution for mobile banking applications, as presented in [2], [8], [9] and soft tokens being used for both log-in and transactions in our target bank [17] thus, we devote the next paragraphs to examine the inner workings and algorithms of OTP generators.

Multiple solutions have been deployed for authenticating clients and servers by various banks, governments and institutions. OATH is the initiative for open authentication. They provide a reference architecture for strong authentication to be applied to any kind of device and network using open standards [62]. They operate in the areas of authentication protocols, credentials for security devices and credential validation. Their methods include public key infrastructure (PKI) [63] and X509 certificates [64] for Internet protocol authentication, OTP credentials for applications depending on user passwords and Subscriber identity module (SIM) [65] for authenticating users over WiFi and cellular networks. OATH provides two OTP algorithms, HOTP and TOTP with the first used to generate OTP codes from Hashed Message Authentication Codes with an increasing counter as input while the latter is a variation of the first one using an integer time counter instead. Both HOTP and TOTP have been adopted by IETF and presented in [67] and [66] consecutively.

According to [67], HOTP is defined within 4 variables C - an 8-bit counter value synchronized between client and server, K - the shared key, T - number of failed authentication attempts before locking the user, s - synchronization parameter that is used to verify codes up to C + s counter values and D - number of digits in the output HOTP code. In the process of calculating the code, first the HMAC-SHA-1 of key and counter is calculated and a 20 byte string is received:

$$HS = HMAC - SHA - 1(K, C)$$

Then, the output string undergoes a dynamic truncation in order reduce the size into a 4 byte string. During the truncation, the low order 4 bits of the final byte in the output string are turned into a number that indicates the $byte\_offset$. At that point, starting with the string position at $byte\_offset + 1$, the next 31 bits are returned. In the final step, the truncated string is converted to a number and with a modulo the number is reduced to the range of $0..10^D - 1$:

$$Num = Str2Num(truncated\_string)$$

$$Return \quad HOTP = Num \bmod 10^D$$

HOTP provides an OTP generation that can be used both by client and sever authentication however, this implementation might have a synchronization problem when more than s codes are generated by one side while no authentication requests are sent on the other side. TOTP comes to solve that problem using an integer time counter instead of an incremental one.

As mentioned earlier, TOTP is a variation of HOTP with the difference being that instead of C, the counter value is generated by time input using three variables: X - time window duration in seconds with default value 30 seconds, T0 - initial Unix time to start counting steps that is by default 0 and T_U - current Unix time. The time counter T is generated as follows:

$$T = floor[(T_U - T0)/X]$$

At this point should be noted that T value must be higher than 32-bits if the system is used after 2038.

Examining an open authentication protocol is rather trivial since everything is given a-priory, on the other hand, vendors may use their own algorithms enforced with obfuscation in order to make it harder to get insight of the algorithms used. In their work, B. Mueller [61] and T. Valverde [68], present how they reverse engineer the soft token OTP generation bypassing all protection layers for RSA SecurID [69], Vasco DIGIPASS [19] and a Brazilian bank.

RSA SecurID operates on AES encryption and protects the code using proGuard obfuscation, making it harder to understand. In addition the app uses SecurIDlib for OTP computation. B. Mueller suggests three starting points for extracting the OTP algorithm:

- Check for classes in SecurIDlib that include keywords related to ”OTP”, ”Signature”, ”token” etc
- Check static plain text strings with keywords related to OTP
- Check for AES S-box or SHA256 algorithm constants

However, in the specific case the procedure becomes simpler with one extra information - stoken [71], is an open source OTP generator compatible with RSA SecurID app. Examining stoken algorithm gives significant insight on how RSA SecurID algorithm operates. Stoken performs 5 rounds of AES encryption using keys that are generated by time-stamps and a PIN code for the final round encryption. , SecurID app uses a class that performs the same algorithm, com.rsa.securidlib.D.n. Furthermore, seed is stored in an SQLite database, securidDB in an encrypted form. At this point the author uses frida [72] and code injection to get the seed plaintext after is decrypted from the app. Finally, using the seed and token serial the OTP is generated through stoken algorithm.

Vasco DIGIPASS is the second case study that B. Mueller provides [61]. It uses in AES and triple DES encryption and supports multiple OTP options based on confirmation codes, challenge-response, MAC signatures and timestamps-events. Furthermore Vasco DIGIPASS employs multiple anti-tampering and anti-debugging mechanisms using libshield.so. This library is responsible for preventing any debugger from attaching to the app, checking whether the device is rooted and kill the app if that is the case and perform checks to make sure that the app is not modified and the APK signature is not altered. In addition, on start-up a child process is attached to the main process as a debugger that monitors and prevents other processes to attach to it. Also, the main process monitors the PIDs of the tasks that trace it and if a PID other than the child's process is found the app is terminated.

To bypass root detection the author hides system calls that attempt to acces root privilege data, creates a fake /proc directory where stat files are not showing any root access as well as a fake /sys/fs/selinux/enforce. Anti-debugging process is dealt by intercepting the first attach attempt from child process and storing its PID while returning an EOK. Then a fake process with child's process PID is shown to main process as the tracer PID. Finally, all trace calls from child process are intercepted and the expected values are returned to the process. After the initial phase of the child process a stop message is snd to put the process to sleep. At this point attaching a debugger to the process is possible but stil the debugger packets are not received due to JdwpState::HandlePacket(). To overcome this the function is changed and the library libart.so updated.

After bypassing the protection layers and performing static and dynamic analysis, the author provides the OTP generation algorithm as follows:

1. Generate $FPH = SHA256(AndroidID + IMEI + Key1)$, Key1 is hard-coded in the app.

2. Generate $DeviceKey = PDBKF2(FPH + Key2, SEED, 320, 32)$, Key2 & seed are hard-coded in the app.

3. Decrypt dynamic and static vectors using DeviceKey and AES-CFB algorithm

4. Final OTP is a function of timestamp, static and dynamic vector data and device fingerprint after being processed by PBKDF2 algorithm.

5. After the OTP is generated dynamic vector updates the number of OTPs generated by one, passes through AES encryption using device key and stored in memory.

Finally, the author presents a proof of concept attack where the java code of the app responsible for generating the OTP is copied and used with the extracted Keys and data.

T. Valverde, in his project [68] he revrese engineers a Brasilian bank in order to make a hardware implementation of OTP generator using a TI Stellaris LaunchPad [73], cryptosuite [74] and RTClib [75]. The bank provides a code generator that is activated through a phone call and is PIN-Protected. For starting, the author extracts the APK and decompiles it using APKtool, dex2jar and JD-GUI, to get the java code of the APK. Then he deobfuscates the exception strings, using hard-coded keys and an algorithm function found in the app. Then by examining the exception strings and classes that are called, he finds out that the OTP generation algorithm is as follows:

4.2.1 Get token Key

A token-key stored in token.db is extracted after decrypting the file using SHA-1 digest of Android_ID as a key.

4.2.2 Generate time-stamp

Time stamp is generated by calculating the number of 36 sec intervals since 1 aprin 2007 00:00.

4.2.3 Generate hash code

Time-stamp becomes a byte array and hashed using HMAC-SHA-1 algorithm with token-key.

4.2.4 Generate code

Last four bits of hashed output are used to generate the code.

4.2.5 Truncate code

Final output code is the modulo 10000 of the generated code.

According to the author, the algorithm of the bank shows high similarities with TOTP Google's authenticatior implementation with small differences in the period that token lasts and time-stamp date.

In this chapter we discussed about authentication mechanisms and the significance of OTP as well as its inner workings. In addition, we have examined two open source OTP generators and three case studies of reverse engineering Android apps with added security layers such as obfuscation and encryption. During the case studies we realize the effort that the organizations put in order to secure their custom OTP generators in order to protect their customers.

In the next chapter is our turn to start the analysis on UK's HSBC bank to reverse engineer the communication between the user and bank in order to provide insight regarding their security.

Chapter 5

HSBC vulnerability assessment

In this chapter, we present the methods used to analyze and assest HSBC online and mobile banking platforms. During the process we discuss some of strong and weak security mechanisms that the banking system employs.

5.1 HSBC mobile banking

HSBC [17] bank in UK provides a mobile banking app that users can download from Google store and use it, either with their devices registered or not. The basic app is used to show balances, bank messages, pay bills and make transfers using Paym system [76], a mobile payment service between seventeen banks and building societies. Registered users link their mobile number to their bank account and using that number they are able to transfer and receive money with a daily transfer money limit.

When someone makes a bank account he may register to online banking and create a username an online banking password as well as a memorable question for log-in purposes. Finally, for verifying the registration he should provide a physical or digital secure secure key. Physical secure key is received by mail and includes a key generating device and an activation code. Digital secure device, refers to the mobile banking app when running on a registered device. To register the device, user should have already registered for online banking and received a validation code for activating digital secure key. By registering a device, user is able to activate digital secure key generator. Using the DSK generator the user is able to generate three kinds of OTPs:

- Log-in OTP
- Confirm transaction OTP
- Register device OTP

Using log-in OTP the user is able to log in to full access online banking that enables making transfers to new and global accounts while confirm transaction

OTP is used to verify such transaction. Finally, when a mobile device is already registered as digital secure key device changing the registration to another device needs the register device OTP otherwise the user should contact the bank and get a secure key by mail. To log in to a registered mobile app user can use his username and a DSK password that he created during registering the device. In the other hand, when user attempts to log-in with a non-registered mobile device as well as online banking he has to provide username, memorable answer and chose between generating a log-in OTP for full access log-in or use part of the online banking password for limited access log-in.

5.2 Starting point - Transferring registration

As shown in the previous section, in HSBC, mobile and online banking are closely cooperating in order to allow the user have a full-access log-in and perform transactions without fully exposing passwords or critical information but providing OTPs instead. Also, for unknown and beyond understanding reasons, mobile banking cannot perform any transactions besides Paym. In our work we aim to analyze the online and mobile banking platforms to first examine the security of mobile app and online interface as well as the client-server communication for log-in, balance view and transactions.

In order to achieve our goals, we have in our hands a registered device and the account related keywords (username, passwords etc.). The registered device we have, has been previously hacked and the APK has a different signature than HSBC's. Furthermore the app is been outdated and does not allow any use if not updated. Since the installed APK is modified and signed by a different key it is not possible to accept an update from HSBC. The only options to get a registered device with an updated APK are:

- Contact the bank and getting a registration activation code by mail.
- Update the device through hacking the app.

The first choice would take time to receive the code and could raise questions by the bank, thus the second approach is decided.

By examining the app files we observed that sharedPrefs folder contained multiple XML shown in figure 5.1 files including the files named SOPValueStore_148 and TOKEN_ValueStore_148. Furthermore, a comparison with an updated non-registered mobile banking app shown that most of the XML files also found in the updated version and are the same in both name and context with the old version, besides some version numbers and dates HSBCHybridSharedPrefs, configuration_148 and nameValueStore files. However, SOTPValueStore_148 is not found in the updated non-registered version thus we considered that it has to do with OTP generation and device registration. For that reason we decided to attempt transferring the files from the registered app to the updated non-registered app to check whether it is possible to transfer the registration just by editing and adding the files from sharedPrefs folders. However this attempt failed thus further examination of the code is required.

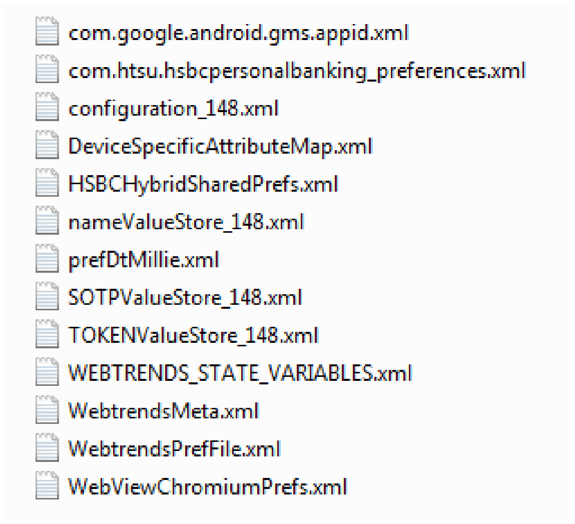

com.google.android.gms.appid.xml
com.htsu.hsbcpersonalbanking_preferences.xml
configuration_148.xml
DeviceSpecificAttributeMap.xml
HSBCHybridSharedPrefs.xml
nameValueStore_148.xml
prefDtMillie.xml
SOTPValueStore_148.xml
TOKENValueStore_148.xml
WEBTRENDS_STATE_VARIABLES.xml
WebtrendsMeta.xml
WebtrendsPrefFile.xml
WebViewChromiumPrefs.xml

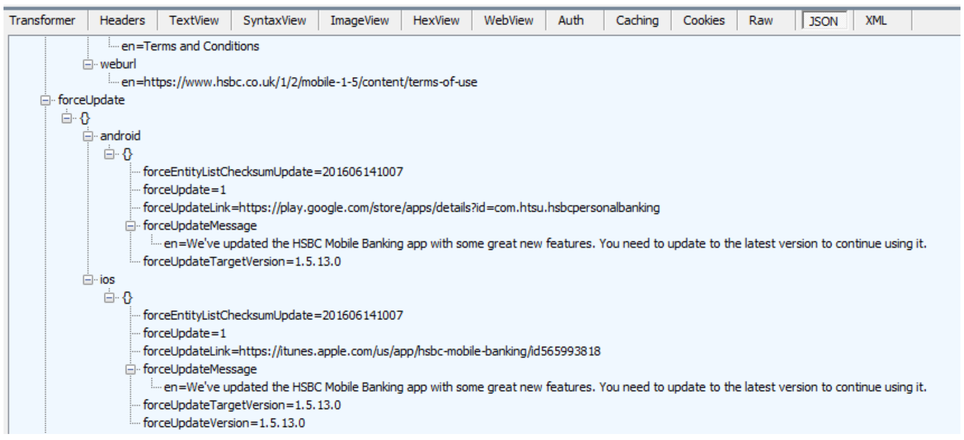
Figure 5.1: SharedPrefs files

In a second test, since the registered device already had certificate checks removed we tried to intercept the communication in order to check whether the update request comes from server or application. Results shown that packet flow is coming normally without any update requests and the app still gives the update message and kills itself afterwards.

In order to find a starting point for hacking the update we examined the java byte codes from both updated and old-registered apps starting with the calls of data from sharedPrefs files and methods related to ”update” or ”register” keywords. At that point several patches were applied to the code without any significant result. However, during our search we came across a forceUpdate form in configuration_148.xml file in sharedPrefs that included the same output string as the one shown on the mobile screen when the app requires an update. The force update form included the following fields under Android section:

- forceUpdate
- forceUpdateTargetVersion
- forceUpdateVersion
- forceUpdateLink
- forceUpdateMessage
- forceEntityListChecksumUpdate

By changing forceUpdateMessage string, the new string is shown on the mobile screen when the update is requested thus it concludes that this form is responsible for the update request. In the next step the forceUpdate value is changed from 1 to 0 and the result is that the application still requires for an update. However, this time the force update request comes from the bank server instead of the application. Specifically, a JSON form is sent containing an updated version of configuration_148.xml with the field of forceUpdate as shown in figure 5.2. At this point, a script is written if fiddler [78] that changes the received JSON form forceUpdate field from 1 to 0 in a same way as we did earlier with the XML file.



```
        en=Terms and Conditions
      weburl
        en=https://www.hsbc.co.uk/1/2/mobile-1-5/content/terms-of-use
  forceUpdate
    {}
      android
        {}
          forceEntityListChecksumUpdate=201606141007
          forceUpdate=1
          forceUpdateLink=https://play.google.com/store/apps/details?id=com.htsu.hsbcpersonalbanking
          forceUpdateMessage
            en=We've updated the HSBC Mobile Banking app with some great new features. You need to update to the latest version to continue using it.
          forceUpdateTargetVersion=1.5.13.0
      ios
        {}
          forceEntityListChecksumUpdate=201606141007
          forceUpdate=1
          forceUpdateLink=https://itunes.apple.com/us/app/hsbc-mobile-banking/id565993818
          forceUpdateMessage
            en=We've updated the HSBC Mobile Banking app with some great new features. You need to update to the latest version to continue using it.
          forceUpdateTargetVersion=1.5.13.0
          forceUpdateVersion=1.5.13.0
```

Figure 5.2: Force update received JSON form.

Having bypassed the forceUpdate message request, updating the app is a series of simple steps:

- Log-in to new device using username
- Provide answers to memorable/security questions
- Provide OTP from registered device
- Use confirmation code received in email
- Create a new DSK password

Since the registration transfer was successful and the required OTP was provided by the outdated app version, is clear that the OTP algorithm is the same as the one used in previous versions. This means that even if the current app uses an enforced security mechanisms to protect their OTP algorithm from being inspected, the algorithm can be extracted simply by using an old and less protected app.

5.3 Exploring Android app

After updating APK and registering the device we attempt to make a backup of the mobile app using adb and extract the APK for analysis. In a first attempt to find out more about the app we patched the Android manifest to make the app debuggable and be able to gain access into sharedPrefs. However, the installed app could not start. In a second attempt we installed the APK without any changes in the code or manifest but with a different APK signature. The result is the same as the first attempt which shows that the app is killed is due to a failure in signature validation.

Doing some further investigation in the APK files we found out that it uses three libraries:

- libmobicheck.so
- libopencv_java.so
- libshield.7cc6b7f1.so

In a first look on the libraries used our attention is on libshield which is also seen in the previous section in B. Mueller's [61] paper. This library is described to perform various tests and employ multiple anti-tampering and anti-debugging methods to protect the app. A check in the Smali code libsield calls, revealed that libshield is called in 293 Smali files. However, in all cases there are only two different function calls that are performed in a single line and return nothing back.

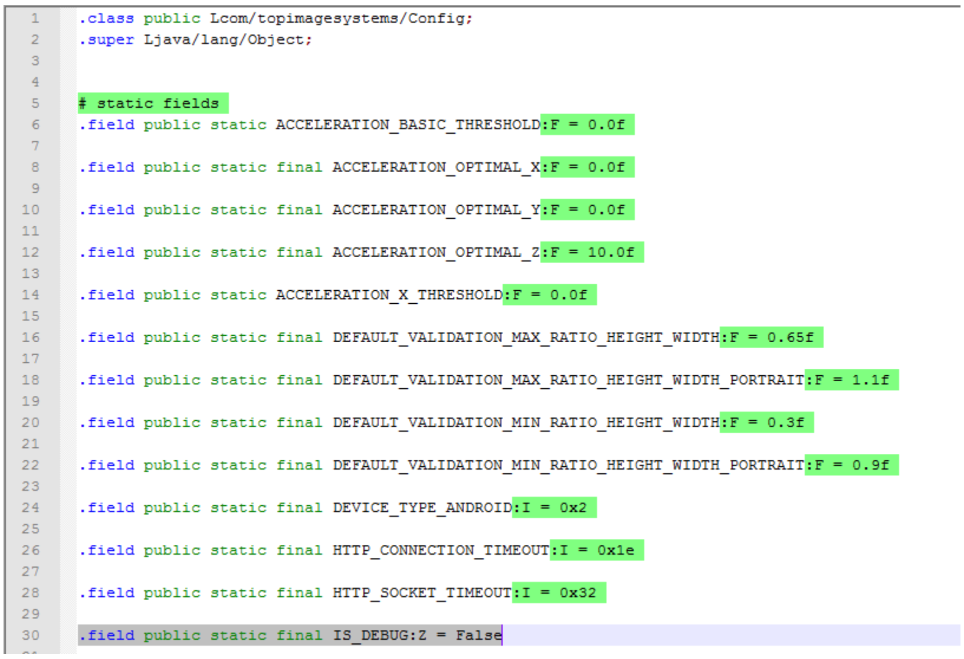
The first line of code that is called in almost every process is
invoke-static , Lno/promon/shield/LibshieldStarter;− >startLibshieldFromClinit()V
The second function call is called only in AppMonitor.Smali and the code is described as
invoke-static p0, Lno/promon/shield/LibshieldStarter;− >
startLibshieldFromAppMonitor(LAndroid/content/Context;)V

Knowing that the libshield function calls do not return anything back, an attempt to remove all function calls, is all that its needed to bypass the signature and debugging protection! To test if the debugging is possible we created a Smali project in intelijIDEA and performed a real time debugging on Android APK monitoring logs from Android DDMS and setting various breakpoints in the code. In a small debugging session it was possible to identify some methods related to SSL certificate protection and are responsible for killing the app when a MITM attack is attempted.

More interestingly, code parts show that debug flags hard-coded in Smali code as shown in figure 5.3. Setting the debug flags to true, more than 600 log messages can be printed within 100 plus Smali files, that probably authors of the app used for debugging. While some logs are used to state that certain app tasks are (should be) taking place when the log is printed, some others provide information regarding the requests and responses that the app is taking. Smali code examples [1] that produces

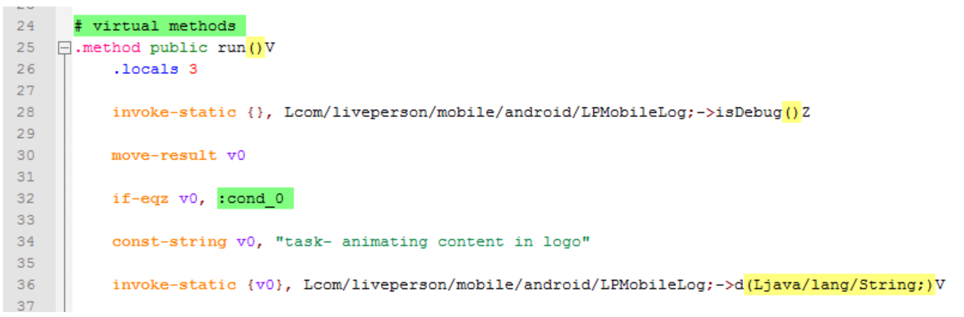
[1]For further understanding of the Smali code refer to [28]

the logs are shown in figure 5.4 and 5.5. However, the significance of these logs might not be the content of the logs but the fact that almost 1/40 th of the total Smali files contains at least one log file. This may provide an important insight on how the code is executed and reduce the work of adding extra logs in the code.



```
 1  .class public Lcom/topimagesystems/Config;
 2  .super Ljava/lang/Object;
 3
 4
 5  # static fields
 6  .field public static ACCELERATION_BASIC_THRESHOLD:F = 0.0f
 7
 8  .field public static final ACCELERATION_OPTIMAL_X:F = 0.0f
 9
10  .field public static final ACCELERATION_OPTIMAL_Y:F = 0.0f
11
12  .field public static final ACCELERATION_OPTIMAL_Z:F = 10.0f
13
14  .field public static ACCELERATION_X_THRESHOLD:F = 0.0f
15
16  .field public static final DEFAULT_VALIDATION_MAX_RATIO_HEIGHT_WIDTH:F = 0.65f
17
18  .field public static final DEFAULT_VALIDATION_MAX_RATIO_HEIGHT_WIDTH_PORTRAIT:F = 1.1f
19
20  .field public static final DEFAULT_VALIDATION_MIN_RATIO_HEIGHT_WIDTH:F = 0.3f
21
22  .field public static final DEFAULT_VALIDATION_MIN_RATIO_HEIGHT_WIDTH_PORTRAIT:F = 0.9f
23
24  .field public static final DEVICE_TYPE_ANDROID:I = 0x2
25
26  .field public static final HTTP_CONNECTION_TIMEOUT:I = 0x1e
27
28  .field public static final HTTP_SOCKET_TIMEOUT:I = 0x32
29
30  .field public static final IS_DEBUG:Z = False
```

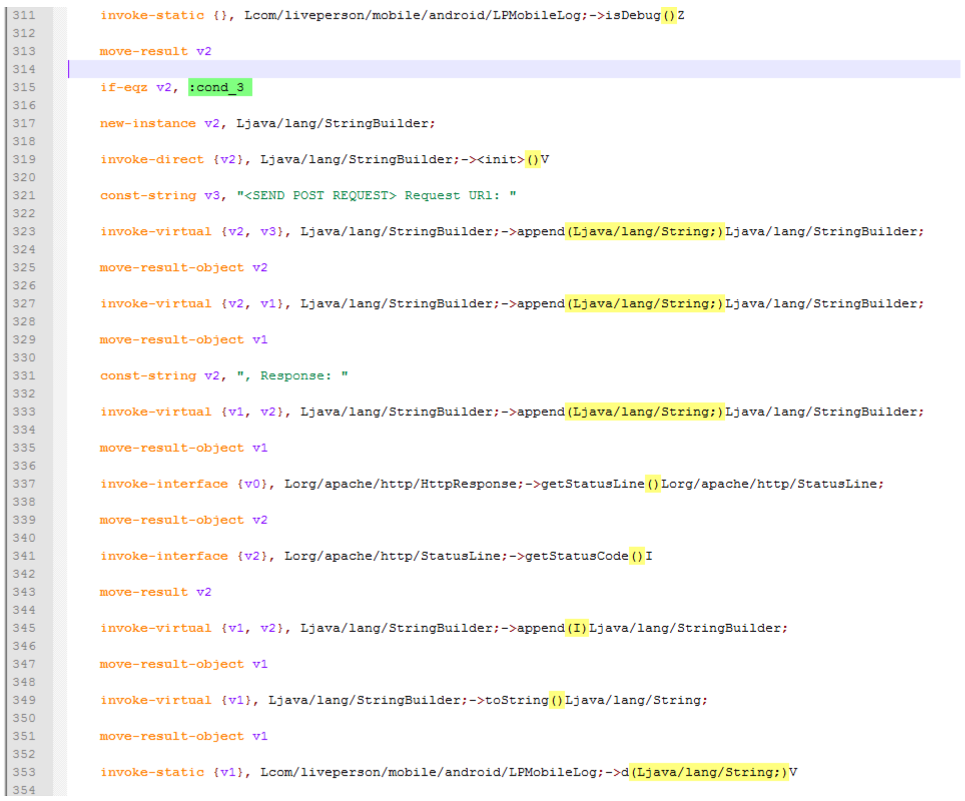
Figure 5.3: Field to enable debugging logs (line 30).



```
24  # virtual methods
25  .method public run()V
26      .locals 3
27
28      invoke-static {}, Lcom/liveperson/mobile/android/LPMobileLog;->isDebug()Z
29
30      move-result v0
31
32      if-eqz v0, :cond_0
33
34      const-string v0, "task- animating content in logo"
35
36      invoke-static {v0}, Lcom/liveperson/mobile/android/LPMobileLog;->d(Ljava/lang/String;)V
37
```

Figure 5.4: The string ”task- animating content in logo” is printed in log (line 36) when isDebug() function returns True.

At this point, since the mobile banking app cannot perform any transactions, further Anrdoid analysis is unnecessary. For that reason, the study moves from Android platform into exploring the other two options for the mobile and online banking, iOS app and web application.



```
invoke-static {}, Lcom/liveperson/mobile/android/LPMobileLog;->isDebug()Z

move-result v2


if-eqz v2, :cond_3

new-instance v2, Ljava/lang/StringBuilder;

invoke-direct {v2}, Ljava/lang/StringBuilder;-><init>()V

const-string v3, "<SEND POST REQUEST> Request URl: "

invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v2

invoke-virtual {v2, v1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

const-string v2, ", Response: "

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

invoke-interface {v0}, Lorg/apache/http/HttpResponse;->getStatusLine()Lorg/apache/http/StatusLine;

move-result-object v2

invoke-interface {v2}, Lorg/apache/http/StatusLine;->getStatusCode()I

move-result v2

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(I)Ljava/lang/StringBuilder;

move-result-object v1

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

move-result-object v1

invoke-static {v1}, Lcom/liveperson/mobile/android/LPMobileLog;->d(Ljava/lang/String;)V
```

Figure 5.5: In the case where isDebug() returns true, the URL that post request was sent will be printed in log along with the response.

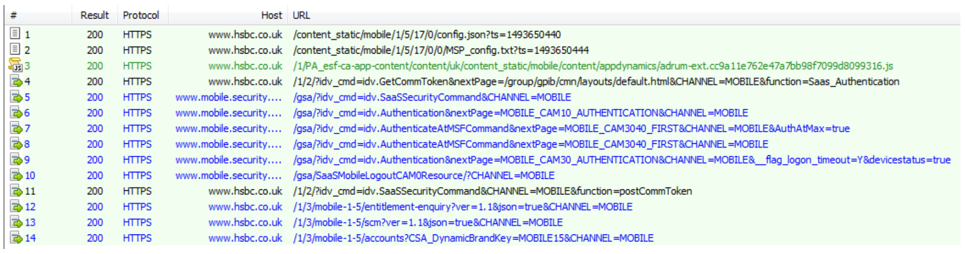
5.4 Man-in-the-Middle in iOS

Since a lot of effort is put into Android app to hack the update/registration and patch it in order to remove app protection layers, in iOS a different approach is tested. A jailbreak iPhone using iOS Kill Switch in order to disable SSL verification functions and a tsprotector to bypass jailbreak detection proved to be more than enough to perform a successful MITM attack capturing communication packets using fiddler. The communication session between iPhone and bank are presented in figure 5.6 for log-in using password and figure 5.7 for log-in using OTP.

The packets captured are used to gain insight regarding inner working of mobile banking app and as a final result, make our own version of the app.

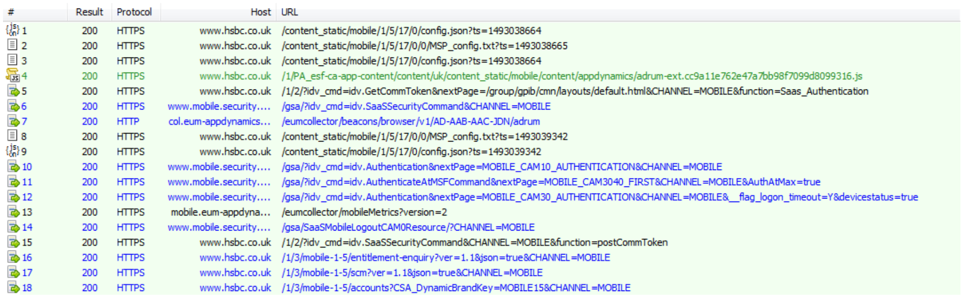
5.5 Man-in-the-Middle in full-access online banking

As mentioned earlier, for unknown reasons, mobile banking app does not let the users perform any transactions besides paym and linked accounts which only permits a limited amount of money to be transferred per day. In order get an insight



#	Result	Protocol	Host	URL
1	200	HTTPS	www.hsbc.co.uk	/content_static/mobile/1/5/17/0/config.json?ts=1493650440
2	200	HTTPS	www.hsbc.co.uk	/content_static/mobile/1/5/17/0/0/MSP_config.txt?ts=1493650444
3	200	HTTPS	www.hsbc.co.uk	/1/PA_esf-ca-app-content/content/uk/content_static/mobile/content/appdynamics/adrum-ext.cc9a11e762e47a7bb98f7099d8099316.js
4	200	HTTPS	www.hsbc.co.uk	/1/2/?idv_cmd=idv.GetCommToken&nextPage=/group/gpib/cmn/layouts/default.html&CHANNEL=MOBILE&function=Saas_Authentication
5	200	HTTPS	www.mobile.security....	/gsa/?idv_cmd=idv.SaaSSecurityCommand&CHANNEL=MOBILE
6	200	HTTPS	www.mobile.security....	/gsa/?idv_cmd=idv.Authentication&nextPage=MOBILE_CAM10_AUTHENTICATION&CHANNEL=MOBILE
7	200	HTTPS	www.mobile.security....	/gsa/?idv_cmd=idv.AuthenticateAtMSFCommand&nextPage=MOBILE_CAM3040_FIRST&CHANNEL=MOBILE&AuthAtMax=true
8	200	HTTPS	www.mobile.security....	/gsa/?idv_cmd=idv.AuthenticateAtMSFCommand&nextPage=MOBILE_CAM3040_FIRST&CHANNEL=MOBILE
9	200	HTTPS	www.mobile.security....	/gsa/?idv_cmd=idv.Authentication&nextPage=MOBILE_CAM30_AUTHENTICATION&CHANNEL=MOBILE&__flag_logon_timeout=Y&devicestatus=true
10	200	HTTPS	www.mobile.security....	/gsa/SaaSMobileLogoutCAM0Resource/?CHANNEL=MOBILE
11	200	HTTPS	www.hsbc.co.uk	/1/2/?idv_cmd=idv.SaaSSecurityCommand&CHANNEL=MOBILE&function=postCommToken
12	200	HTTPS	www.hsbc.co.uk	/1/3/mobile-1-5/entitlement-enquiry?ver=1.1&json=true&CHANNEL=MOBILE
13	200	HTTPS	www.hsbc.co.uk	/1/3/mobile-1-5/scm?ver=1.1&json=true&CHANNEL=MOBILE
14	200	HTTPS	www.hsbc.co.uk	/1/3/mobile-1-5/accounts?CSA_DynamicBrandKey=MOBILE15&CHANNEL=MOBILE

Figure 5.6: Log-in session for iPhone using Password



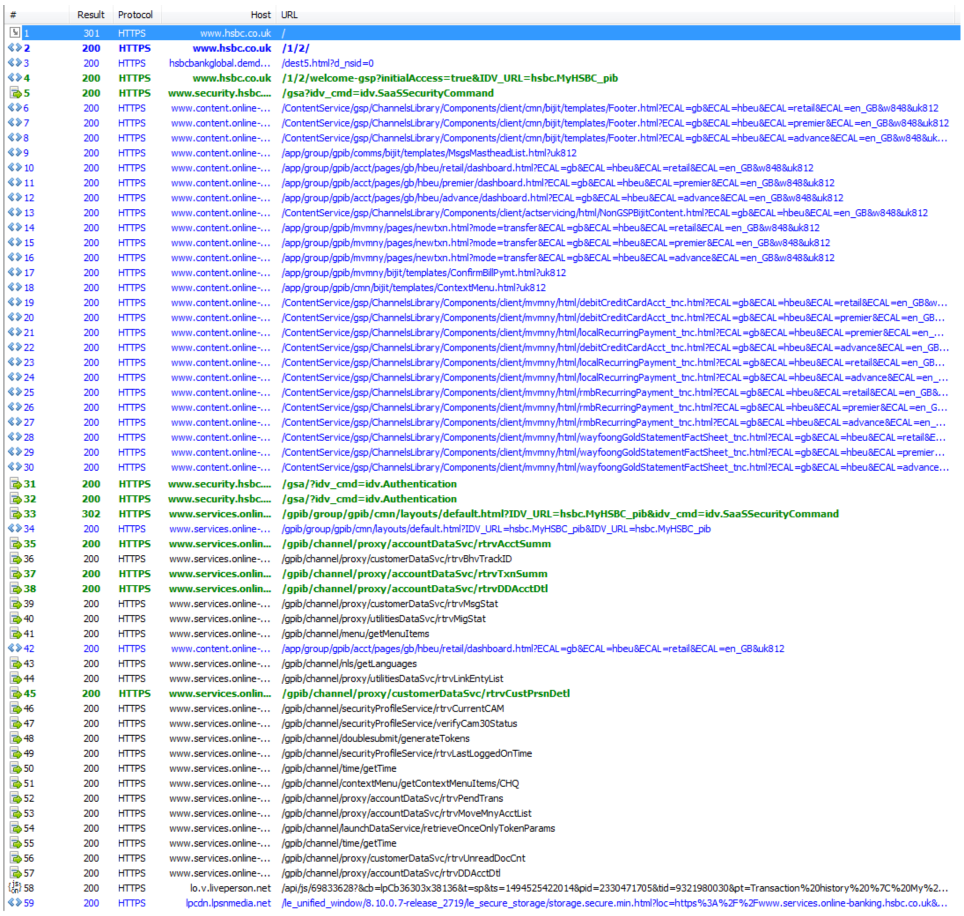
#	Result	Protocol	Host	URL
1	200	HTTPS	www.hsbc.co.uk	/content_static/mobile/1/5/17/0/config.json?ts=1493038664
2	200	HTTPS	www.hsbc.co.uk	/content_static/mobile/1/5/17/0/0/MSP_config.txt?ts=1493038665
3	200	HTTPS	www.hsbc.co.uk	/content_static/mobile/1/5/17/0/config.json?ts=1493038664
4	200	HTTPS	www.hsbc.co.uk	/1/PA_esf-ca-app-content/content/uk/content_static/mobile/content/appdynamics/adrum-ext.cc9a11e762e47a7bb98f7099d8099316.js
5	200	HTTPS	www.hsbc.co.uk	/1/2/?idv_cmd=idv.GetCommToken&nextPage=/group/gpib/cmn/layouts/default.html&CHANNEL=MOBILE&function=Saas_Authentication
6	200	HTTPS	www.mobile.security....	/gsa/?idv_cmd=idv.SaaSSecurityCommand&CHANNEL=MOBILE
7	200	HTTP	col.eum-appdynamics...	/eumcollector/beacons/browser/v1/AD-AAB-AAC-JDN/adrum
8	200	HTTPS	www.hsbc.co.uk	/content_static/mobile/1/5/17/0/0/MSP_config.txt?ts=1493039342
9	200	HTTPS	www.hsbc.co.uk	/content_static/mobile/1/5/17/0/config.json?ts=1493039342
10	200	HTTPS	www.mobile.security....	/gsa/?idv_cmd=idv.Authentication&nextPage=MOBILE_CAM10_AUTHENTICATION&CHANNEL=MOBILE
11	200	HTTPS	www.mobile.security....	/gsa/?idv_cmd=idv.AuthenticateAtMSFCommand&nextPage=MOBILE_CAM3040_FIRST&CHANNEL=MOBILE&AuthAtMax=true
12	200	HTTPS	www.mobile.security....	/gsa/?idv_cmd=idv.Authentication&nextPage=MOBILE_CAM30_AUTHENTICATION&CHANNEL=MOBILE&__flag_logon_timeout=Y&devicestatus=true
13	200	HTTPS	mobile.eum-appdyna...	/eumcollector/mobileMetrics?version=2
14	200	HTTPS	www.mobile.security....	/gsa/SaaSMobileLogoutCAM0Resource/?CHANNEL=MOBILE
15	200	HTTPS	www.hsbc.co.uk	/1/2/?idv_cmd=idv.SaaSSecurityCommand&CHANNEL=MOBILE&function=postCommToken
16	200	HTTPS	www.hsbc.co.uk	/1/3/mobile-1-5/entitlement-enquiry?ver=1.1&json=true&CHANNEL=MOBILE
17	200	HTTPS	www.hsbc.co.uk	/1/3/mobile-1-5/scm?ver=1.1&json=true&CHANNEL=MOBILE
18	200	HTTPS	www.hsbc.co.uk	/1/3/mobile-1-5/accounts?CSA_DynamicBrandKey=MOBILE15&CHANNEL=MOBILE

Figure 5.7: Log-in session for iPhone using OTP

of the transactions functionality, we had to work with the online banking as well.

Online banking provides two options to the user: 1- limited-access banking where the user can log-in providing username, memorable answer and some requested password digits, 2- full-access banking in which the user can log-in through username, memorable answer and OTP generated by a registered device. In the limited access log-in user can use online banking services but he can not perform transactions to accounts that he has never sent money before while full-access online banking has no limit on where the user can send money to.

Working with online banking proved to be even simpler than iOS since there are no protection mechanisms between client and server and a MITM attack can be perform just by redirecting the packets to fiddler. In figure 5.8, the packet exchange during log-in phase is shown. comparing figures 5.7 and 5.8, is concluded that web application sends almost 3 times more packets just for log-in.



#	Result	Protocol	Host	URL
1	301	HTTPS	www.hsbc.co.uk	/
2	**200**	**HTTPS**	**www.hsbc.co.uk**	**/1/2/**
3	200	HTTPS	hsbcbankglobal.demd...	/dest5.html?d_nsid=0
4	**200**	**HTTPS**	**www.hsbc.co.uk**	**/1/2/welcome-gsp?initialAccess=true&IDV_URL=hsbc.MyHSBC_pib**
5	**200**	**HTTPS**	**www.security.hsbc....**	**/gsa?idv_cmd=idv.SaaSSecurityCommand**
6	200	HTTPS	www.content.online-...	/ContentService/gsp/ChannelsLibrary/Components/client/cmn/bijit/templates/Footer.html?ECAL=gb&ECAL=hbeu&ECAL=retail&ECAL=en_GB&w848&uk812
7	200	HTTPS	www.content.online-...	/ContentService/gsp/ChannelsLibrary/Components/client/cmn/bijit/templates/Footer.html?ECAL=gb&ECAL=hbeu&ECAL=premier&ECAL=en_GB&w848&uk812
8	200	HTTPS	www.content.online-...	/ContentService/gsp/ChannelsLibrary/Components/client/cmn/bijit/templates/Footer.html?ECAL=gb&ECAL=hbeu&ECAL=advance&ECAL=en_GB&w848&uk...
9	200	HTTPS	www.content.online-...	/app/group/gpib/comms/bijit/templates/MsgsMastheadList.html?uk812
10	200	HTTPS	www.content.online-...	/app/group/gpib/acct/pages/gb/hbeu/retail/dashboard.html?ECAL=gb&ECAL=hbeu&ECAL=retail&ECAL=en_GB&w848&uk812
11	200	HTTPS	www.content.online-...	/app/group/gpib/acct/pages/gb/hbeu/premier/dashboard.html?ECAL=gb&ECAL=hbeu&ECAL=premier&ECAL=en_GB&w848&uk812
12	200	HTTPS	www.content.online-...	/app/group/gpib/acct/pages/gb/hbeu/advance/dashboard.html?ECAL=gb&ECAL=hbeu&ECAL=advance&ECAL=en_GB&w848&uk812
13	200	HTTPS	www.content.online-...	/ContentService/gsp/ChannelsLibrary/Components/client/actservicing/html/NonGSPBijitContent.html?ECAL=gb&ECAL=hbeu&ECAL=en_GB&w848&uk812
14	200	HTTPS	www.content.online-...	/app/group/gpib/mvmny/pages/newtxn.html?mode=transfer&ECAL=gb&ECAL=hbeu&ECAL=retail&ECAL=en_GB&w848&uk812
15	200	HTTPS	www.content.online-...	/app/group/gpib/mvmny/pages/newtxn.html?mode=transfer&ECAL=gb&ECAL=hbeu&ECAL=premier&ECAL=en_GB&w848&uk812
16	200	HTTPS	www.content.online-...	/app/group/gpib/mvmny/pages/newtxn.html?mode=transfer&ECAL=gb&ECAL=hbeu&ECAL=advance&ECAL=en_GB&w848&uk812
17	200	HTTPS	www.content.online-...	/app/group/gpib/mvmny/bijit/templates/ConfirmBillPymt.html?uk812
18	200	HTTPS	www.content.online-...	/app/group/gpib/cmn/bijit/templates/ContextMenu.html?uk812
19	200	HTTPS	www.content.online-...	/ContentService/gsp/ChannelsLibrary/Components/client/mvmny/html/debitCreditCardAcct_tnc.html?ECAL=gb&ECAL=hbeu&ECAL=retail&ECAL=en_GB&w...
20	200	HTTPS	www.content.online-...	/ContentService/gsp/ChannelsLibrary/Components/client/mvmny/html/debitCreditCardAcct_tnc.html?ECAL=gb&ECAL=hbeu&ECAL=premier&ECAL=en_GB...
21	200	HTTPS	www.content.online-...	/ContentService/gsp/ChannelsLibrary/Components/client/mvmny/html/localRecurringPayment_tnc.html?ECAL=gb&ECAL=hbeu&ECAL=premier&ECAL=en_...
22	200	HTTPS	www.content.online-...	/ContentService/gsp/ChannelsLibrary/Components/client/mvmny/html/debitCreditCardAcct_tnc.html?ECAL=gb&ECAL=hbeu&ECAL=advance&ECAL=en_GB...
23	200	HTTPS	www.content.online-...	/ContentService/gsp/ChannelsLibrary/Components/client/mvmny/html/localRecurringPayment_tnc.html?ECAL=gb&ECAL=hbeu&ECAL=retail&ECAL=en_GB...
24	200	HTTPS	www.content.online-...	/ContentService/gsp/ChannelsLibrary/Components/client/mvmny/html/localRecurringPayment_tnc.html?ECAL=gb&ECAL=hbeu&ECAL=advance&ECAL=en_...
25	200	HTTPS	www.content.online-...	/ContentService/gsp/ChannelsLibrary/Components/client/mvmny/html/rmbRecurringPayment_tnc.html?ECAL=gb&ECAL=hbeu&ECAL=retail&ECAL=en_GB&...
26	200	HTTPS	www.content.online-...	/ContentService/gsp/ChannelsLibrary/Components/client/mvmny/html/rmbRecurringPayment_tnc.html?ECAL=gb&ECAL=hbeu&ECAL=premier&ECAL=en_G...
27	200	HTTPS	www.content.online-...	/ContentService/gsp/ChannelsLibrary/Components/client/mvmny/html/rmbRecurringPayment_tnc.html?ECAL=gb&ECAL=hbeu&ECAL=advance&ECAL=en_...
28	200	HTTPS	www.content.online-...	/ContentService/gsp/ChannelsLibrary/Components/client/mvmny/html/wayfoongGoldStatementFactSheet_tnc.html?ECAL=gb&ECAL=hbeu&ECAL=retail&E...
29	200	HTTPS	www.content.online-...	/ContentService/gsp/ChannelsLibrary/Components/client/mvmny/html/wayfoongGoldStatementFactSheet_tnc.html?ECAL=gb&ECAL=hbeu&ECAL=premier...
30	200	HTTPS	www.content.online-...	/ContentService/gsp/ChannelsLibrary/Components/client/mvmny/html/wayfoongGoldStatementFactSheet_tnc.html?ECAL=gb&ECAL=hbeu&ECAL=advance...
31	**200**	**HTTPS**	**www.security.hsbc....**	**/gsa/?idv_cmd=idv.Authentication**
32	**200**	**HTTPS**	**www.security.hsbc....**	**/gsa/?idv_cmd=idv.Authentication**
33	**302**	**HTTPS**	**www.services.onlin...**	**/gpib/group/gpib/cmn/layouts/default.html?IDV_URL=hsbc.MyHSBC_pib&idv_cmd=idv.SaaSSecurityCommand**
34	200	HTTPS	www.services.online-...	/gpib/group/gpib/cmn/layouts/default.html?IDV_URL=hsbc.MyHSBC_pib&IDV_URL=hsbc.MyHSBC_pib
35	**200**	**HTTPS**	**www.services.onlin...**	**/gpib/channel/proxy/accountDataSvc/rtrvAcctSumm**
36	200	HTTPS	www.services.online-...	/gpib/channel/proxy/customerDataSvc/rtrvBhvTrackID
37	**200**	**HTTPS**	**www.services.onlin...**	**/gpib/channel/proxy/accountDataSvc/rtrvTxnSumm**
38	**200**	**HTTPS**	**www.services.onlin...**	**/gpib/channel/proxy/accountDataSvc/rtrvDDAcctDtl**
39	200	HTTPS	www.services.online-...	/gpib/channel/proxy/customerDataSvc/rtrvMsgStat
40	200	HTTPS	www.services.online-...	/gpib/channel/proxy/utilitiesDataSvc/rtrvMigStat
41	200	HTTPS	www.services.online-...	/gpib/channel/menu/getMenuItems
42	200	HTTPS	www.content.online-...	/app/group/gpib/acct/pages/gb/hbeu/retail/dashboard.html?ECAL=gb&ECAL=hbeu&ECAL=retail&ECAL=en_GB&uk812
43	200	HTTPS	www.services.online-...	/gpib/channel/nls/getLanguages
44	200	HTTPS	www.services.online-...	/gpib/channel/proxy/utilitiesDataSvc/rtrvLinkEntyList
45	**200**	**HTTPS**	**www.services.onlin...**	**/gpib/channel/proxy/customerDataSvc/rtrvCustPrsnDetl**
46	200	HTTPS	www.services.online-...	/gpib/channel/securityProfileService/rtrvCurrentCAM
47	200	HTTPS	www.services.online-...	/gpib/channel/securityProfileService/verifyCam30Status
48	200	HTTPS	www.services.online-...	/gpib/channel/doublesubmit/generateTokens
49	200	HTTPS	www.services.online-...	/gpib/channel/securityProfileService/rtrvLastLoggedOnTime
50	200	HTTPS	www.services.online-...	/gpib/channel/time/getTime
51	200	HTTPS	www.services.online-...	/gpib/channel/contextMenu/getContextMenuItems/CHQ
52	200	HTTPS	www.services.online-...	/gpib/channel/proxy/accountDataSvc/rtrvPendTrans
53	200	HTTPS	www.services.online-...	/gpib/channel/proxy/accountDataSvc/rtrvMoveMnyAcctList
54	200	HTTPS	www.services.online-...	/gpib/channel/launchDataService/retrieveOnceOnlyTokenParams
55	200	HTTPS	www.services.online-...	/gpib/channel/time/getTime
56	200	HTTPS	www.services.online-...	/gpib/channel/proxy/customerDataSvc/rtrvUnreadDocCnt
57	200	HTTPS	www.services.online-...	/gpib/channel/proxy/accountDataSvc/rtrvDDAcctDtl
58	200	HTTPS	lo.v.liveperson.net	/api/js/69833628?&cb=lpCb36303x38136&t=sp&ts=1494525422014&pid=2330471705&tid=9321980030&pt=Transaction%20history%20%7C%20My%2...
59	200	HTTPS	lpcdn.lpsnmedia.net	/le_unified_window/8.10.0.7-release_2719/le_secure_storage/storage.secure.min.html?loc=https%3A%2F%2Fwww.services.online-banking.hsbc.co.uk&...

Figure 5.8: Log-in session for iPhone

Chapter 6

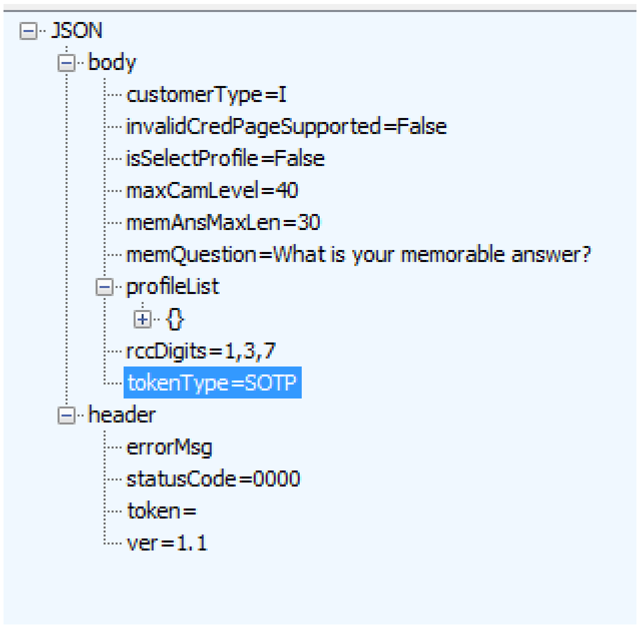
HSBC mobile & online banking

Performing a successful MITM between application and bank service, we are able to capture all communication that happens in between. Furthermore, using fiddler's features, enables us to process the sessions for deeper analysis and extract the desired information. Gaining access to the communication sessions for both mobile and online banking we analyze the following functions:

- Mobile banking: log-in using Password
- Mobile banking: log-in using OTP
- Mobile banking: balance view
- Mobile banking: transaction
- Mobile banking: log-out
- Online banking: log-in using OTP
- Online banking: balance view
- Online banking: transaction history
- Online banking: view known accounts
- Online banking: transaction to known European accounts
- Online banking: transaction to new European accounts

6.1 Mobile banking walk-through

In mobile banking, log-in process using OTP or password is quite similar. Log-in through password is the simplest way of log-in in the mobile banking app since no OTP generator device is needed and in both log-in types the user cannot perform any transactions besides paym and linked accounts.



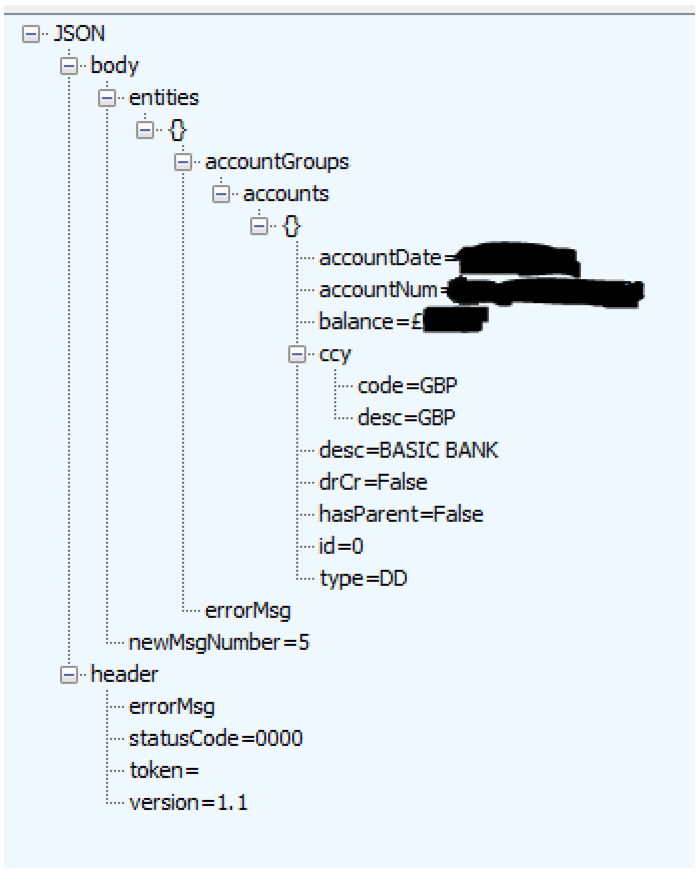
```
JSON
  body
    customerType=I
    invalidCredPageSupported=False
    isSelectProfile=False
    maxCamLevel=40
    memAnsMaxLen=30
    memQuestion=What is your memorable answer?
    profileList
      {}
    rccDigits=1,3,7
    tokenType=SOTP
  header
    errorMsg
    statusCode=0000
    token=
    ver=1.1
```

Figure 6.1: Mobile banking log-in questions. rccDigits refers to the digits of password that the user must provide, in that case 1st 3rd and 7th.

The log-in session starts with three GET requests sent to www.hsbc.co.uk for getting the app related content. Then a POST request is made to start software as a service (SAAS) authentication and in return, SAAS token ID and SAAS token assertion ID are produced and sent in a response along with a session cookie - JSESSIONID(1) and the URL, www.security.hsbc.co.uk to the user.

In the next step, a POST request is sent to the new URL containing SAAS token ID and SAAS token assertion ID. In reply to that, a status code showing that the SAAS token ID and SAAS token assertion ID are valid and two new session cookies -JSESSIONID(2,3) are received. For the rest of the communication with www.mobile.security.hsbc.co.uk the JSESSIONID(3) is used. Being identified by the host, in the next session the client is required POST its userID and receives in reply to provide his memorable answer and either three digits from his password or an OTP as shown in figure 6.1. After that a new POST request is made to initiate OTP log-in. If the user wants to log-in through password another POST request should be sent for selecting password log-in.

In the case where the user chooses to log-in using his password, he has to provide the three requested digits of his password along with his memorable answer and if he chooses the OTP log-in he has to provide a generated OTP using his registered device along with his memorable answer. For both requests a reply is sent noting that log-in is successful and including the last successful log-on date and time. When using a registered device for log-in OTP generation, user has to provide the Finally, a new POST request is sent to redirect the user back to the initial URL with



```
JSON
  body
    entities
      {}
        accountGroups
          accounts
            {}
              accountDate=[redacted]
              accountNum=[redacted]
              balance=£[redacted]
              ccy
                code=GBP
                desc=GBP
              desc=BASIC BANK
              drCr=False
              hasParent=False
              id=0
              type=DD
        errorMsg
    newMsgNumber=5
  header
    errorMsg
    statusCode=0000
    token=
    version=1.1
```

Figure 6.2: Mobile banking balance view.

a new SAAS token ID and SAAS token assertion ID. Using the provided SAAS token assertion ID, SAAS token ID and JSESSIONID(1) cookie the user verifies his authentication and the log-in process is completed.

After a successful log-in the user can view his accounts and balances by sending a single POST request. The balances are presented in a JSON form as shown on figure 6.2.

A money tranfer can be performed using the app through linked accounts. This is possible after log-in by sending a POST request including a form with a field ”requestName = prepare_inhouse_transfer”. However, as a reply we received a json form with the followig description field: ”desc=Sorry, you do not have any UK accounts that will allow you to make a domestic payment or transfer using this app. Please select Global Transfers from the menu to move money between your globally linked accounts. (PMT022)”.

Finally, log-out takes place sending a single POST request with the command log-off in a JSON form.

6.2 Online banking walk-through

Online banking log-in follows a similar process to mobile banking log-in with some variations. At first GET requests to HSBC.co.uk are sent in order to get the web page content. Within the GET request content, SAAS token ID and SAAS token assertion ID are included along with the field action='`https://www.security.hsbc.co.uk/gsa?idv_cmd=idv.SaaSSecurityCommand`'. Then a POST request is sent to www.security.hsbc.co.uk to initiate SAAS authentication followed by multiple GET requests in order to obtain the page content. Similar to the mobile banking log-in, the first step user sends his userID in a post request and then he is required to provide memorable answer and OTP to proceed with full access log-in. There is also the option for limited access log-in where the user provides parts of his password that is not included in our research. Along with log-in information in both userID and password/memorable-answer POST requests, the user sends a field called ESDS_token-null<number> along with the value 0. That token is received from the previous session reply and <number> is different every time. In password/memorable answer POST request, if the log-in is successful the user will receive a new SAAS token ID and SAAS assertion token ID. Finally, the obtained tokens will be sent to www.services.online-banking.hsbc.co.uk in a POST request and a session cookie JSESSIONID will be received. By obtaining the JSESSIONID log-in process is complete.

After a successful log-in, a POST request is sent in order to obtain a cookie named SYNC_TOKEN that should be included in all future sessions in the header field "X-HDR-Synchronizer-Token" along with JSESSIONID. From that point forward a series of POST request and replies are taking place in order perform various functions and retrieve account information. The most significant requests are selected and presented in table 6.1. Before moving to transactions and other web application functions, it should be noted that in online banking, all responses from the bank up to this point are in HTML code in contrast to mobile banking app that all log-in process is done by exchanging JSON forms. However, after a successful log-in, online banking replies include a JSON form with all the requested fields and values.

Using the commands from table 6.1, the user is able to retrieve information about balances, transaction history and even his personal data. Furthermore, the user can perform money transfers to known accounts just by selecting the account that he wants to send the money to or to new accounts by filling a long form.

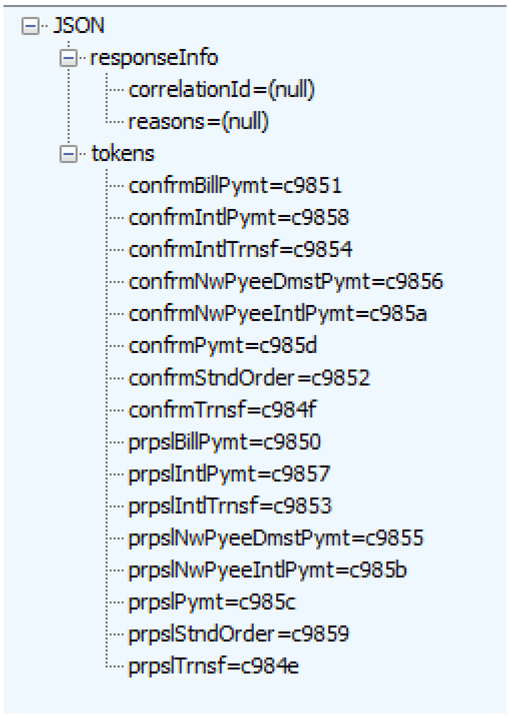
The packet exchange during a transfer to a known account is presented in figure 6.4, with strikeout sessions being identified as redundant packets. As shown from the figure more than half of the sessions taking place during a transaction are not needed. A transaction starts with a POST request sending a single field form to initiate a new payment. Then another POST request is sent asking for operation codes for each transaction type and its confirmation code. These codes are different every time a new transaction is requested and are used by the user as a code in the header field "X-HDR-DoubleSubmit-Token". An example of the received

Function name	Description
rtrvAcctSumm	User bank accounts basic information: Balane, account holder name account index and ID
rtrvTxnSumm	Transaction history per bank account
rtrvDDAcctDtl	User bank accounts detailed information: IBAN info, last update, limit etc.
rtrvMsgStat	Number of bank messages that the user has
menu/getMenuItems rtrvCustPrsnDetl	Menu optons User personal details: Name, address, phone, email, date of birth
rtrvLastLoggedOnTime	Last log on time
rtrvPendTrans	List of pending transactions
rtrvMoveMnyAcctList	List of accounts eligible for moving money
getTime	Server time
rtrvUnreadDocCnt	Numbe of unread bank documents

Table 6.1: List of requests that are sent after log-in to retrieve information and their description.

tokens is shown in figure 6.3. After obtaining the tokens a request is performed to retrieve a list of accounts that are eligible to money transfer and their account index and following that another request to retrieve a list with all known payees. Known payees are the accounts that the user has made a transfer to in the past. Those two sessions can be avoided since after log-in, the exact same requests are performed and the information is already obtained by the user. In the next session, another single field form is being posted to the server to declare a new online banking transfer (OTR) followed by the payment details. For this request, the header "X-HDR-DoubleSubmit-Token" will take the value of the code corresponding to "prpslPymt" token. In the response a payeeID will be included which is used for the next and final request in order to confirm the payment. Payment confirmation request includes the information that was sent in the previous request, payeeID and the header field "X-HDR-DoubleSubmit-Token" gets the code value of "confirmPymt" token. At this point should be noted that transaction to a known account does not need an OTP to be completed.

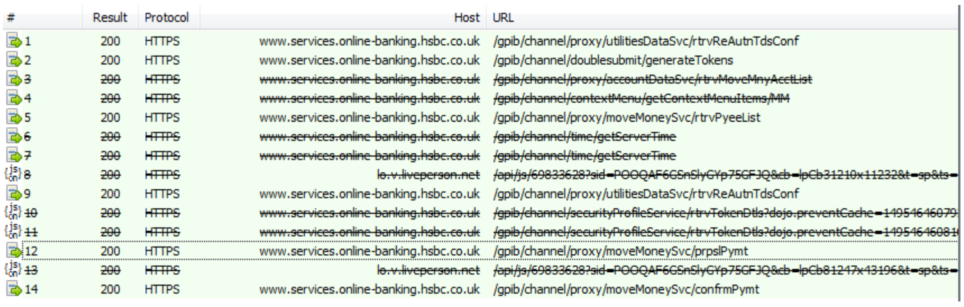
Money transfer to a new account is almost identical shares a lot of similarities with the known account transfer however, some extra steps/sessions are involved in the procedure. Before analyzing log-in procedure should be noted that the following applies to transfers to European banks, and is being tested using a Dutch bank ABN AMRO and UK's bank Barkleys. It starts with the tokens POST request and again the lists of known payees and accounts that are used for transfers are requested. After the lists and tokens are obtained a POST request for OTR is sent followed by a request to note that the transfer is to a new account. Following that,



```
JSON
  responseInfo
    correlationId=(null)
    reasons=(null)
  tokens
    confrmBillPymt=c9851
    confrmIntlPymt=c9858
    confrmIntlTrnsf=c9854
    confrmNwPyeeDmstPymt=c9856
    confrmNwPyeeIntlPymt=c985a
    confrmPymt=c985d
    confrmStndOrder=c9852
    confrmTrnsf=c984f
    prpslBillPymt=c9850
    prpslIntlPymt=c9857
    prpslIntlTrnsf=c9853
    prpslNwPyeeDmstPymt=c9855
    prpslNwPyeeIntlPymt=c985b
    prpslPymt=c985c
    prpslStndOrder=c9859
    prpslTrnsf=c984e
```

Figure 6.3: Tokens different for transfer and confirmation operation.

the user must provide the information of the account that he is sending the money to, along with the personal information of the one receiving them. Then a POST request is sent containing a JSON form with account number and country code in order to receive a list of possible BIC codes and bank details. The user will chose the bank details and BIC code from the list provided. Then, user will choose the currency of transfer and if is not GBP another POST request is sent in order to receive the amount in GBP and the selected currency. At this point, the information is combined to fill a JSON form and a POST request is sent for making the payment. For the transfer request, the header field ”X-HDR-DoubleSubmit-Token” gets the code value of ”prpslNwPyeeIntPymt” token and an OTP is required. For generating the OTP, DSK generator gets as input the last four digits of the receiver’s IBAN and the DSK password. In the bank’s response a form is sent that includes the posted transfer information (Payee account, personal info, transfer account, amount in selected currencies), the transfer fee charge and a transaction ID. Finally, the user will POST a confirmation form with the transfer information and transaction ID. The full procedure is presented in figure 6.5 with strikeout sessions being redundant and red ones being cancelled in fiddler to avoid the charges from bank fees.



#	Result	Protocol	Host	URL
1	200	HTTPS	www.services.online-banking.hsbc.co.uk	/gpib/channel/proxy/utilitiesDataSvc/rtrvReAutnTdsConf
2	200	HTTPS	www.services.online-banking.hsbc.co.uk	/gpib/channel/doublesubmit/generateTokens
~~3~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/proxy/accountDataSvc/rtrvMoveMnyAcctList~~
~~4~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/contextMenu/getContextMenuItems/MM~~
5	200	HTTPS	www.services.online-banking.hsbc.co.uk	/gpib/channel/proxy/moveMoneySvc/rtrvPyeeList
~~6~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/time/getServerTime~~
~~7~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/time/getServerTime~~
~~8~~	~~200~~	~~HTTPS~~	~~lo.v.liveperson.net~~	~~/api/js/69833628?sid=POOQAF6GSnSlyGYp75GFJQ&cb=lpCb31210x11232&t=sp&ts=~~
9	200	HTTPS	www.services.online-banking.hsbc.co.uk	/gpib/channel/proxy/utilitiesDataSvc/rtrvReAutnTdsConf
~~10~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/securityProfileService/rtrvTokenDtls?dojo.preventCache=14954646079~~
~~11~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/securityProfileService/rtrvTokenDtls?dojo.preventCache=14954646081~~
12	200	HTTPS	www.services.online-banking.hsbc.co.uk	/gpib/channel/proxy/moveMoneySvc/prpslPymt
~~13~~	~~200~~	~~HTTPS~~	~~lo.v.liveperson.net~~	~~/api/js/69833628?sid=POOQAF6GSnSlyGYp75GFJQ&cb=lpCb81247x43196&t=sp&ts=~~
14	200	HTTPS	www.services.online-banking.hsbc.co.uk	/gpib/channel/proxy/moveMoneySvc/confrmPymt

Figure 6.4: User-bank communication for money transfer to known accounts.

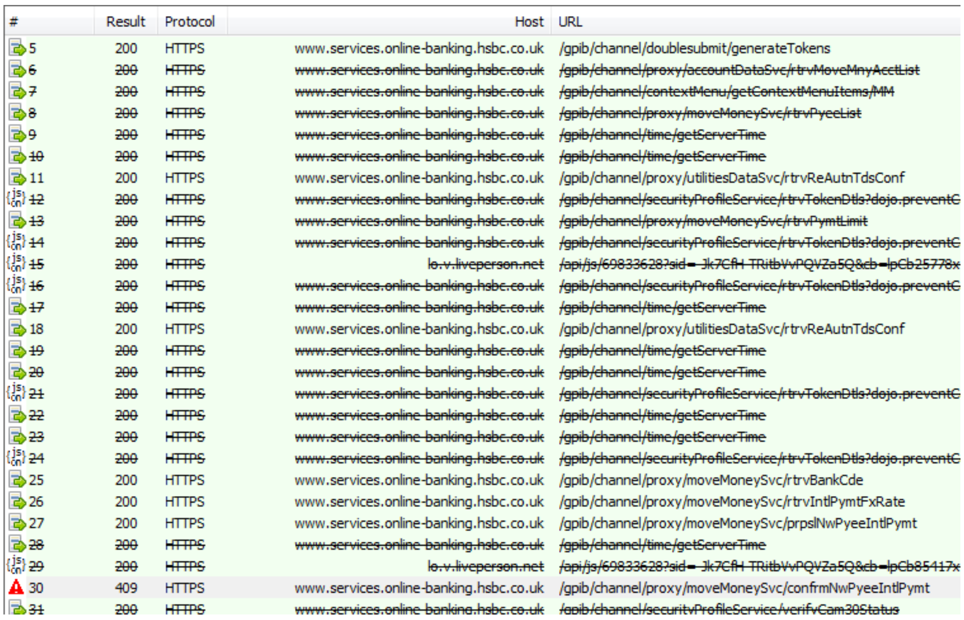
6.3 Security assessment

One of our first concerns using the mobile/online banking services lies around suspicious activity. While working with the app, we observed that the bank allows users to log-in multiple successive times, without a proper log out from the last log-on time. In addition, log-on through different devices at the same time, seems to be also possible, since multiple times in our tests two or even three devices were connected to the mobile app or online banking.

Examining HSBC's inner-workings and communications we see that even though a powerful app is developed, protected by multiple security layers, the functions for OTP generation are still the same with their older, outdated apps. Furthermore, their choice of limiting their well secured mobile banking app from performing transactions, while letting transactions being done in the online banking web service, which is vulnerable to MITM attacks, might not be wise. Adding to that, a successful MITM attack on the online banking not only exposes part of the password and the memorable answer to the attacker during a log-on, but also account details, history, balances and even user identity including name, address, phone and even date of birth.

Besides data theft, MITM attack can also result to even worse consequences when the victim attempts a transaction. When a transaction to a known payee takes place, the attacker will receive the known payee list and by altering the packet data, he can redirect the money to other known accounts or change the amount that is transferred. On the other hand, redirecting the money to a new account is not possible since for transaction to unknown accounts the OTP generation demands adding the last 4 digits of the account. However, in the case were a transaction to a new payee takes place, the attacker could still change the amount of money that is transferred.

Summing up, with a MITM attack in online banking, an attacker could obtain the personal information of the user, his bank account details, history and is even able to alter the amount of money transferred when a new transaction takes place. Adding to this, if an attacker is able to convince a victim to transfer a small amount of money then he could alter that amount without the victim noticing.



#	Result	Protocol	Host	URL
5	200	HTTPS	www.services.online-banking.hsbc.co.uk	/gpib/channel/doublesubmit/generateTokens
~~6~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/proxy/accountDataSvc/rtrvMoveMnyAcctList~~
~~7~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/contextMenu/getContextMenuItems/MM~~
~~8~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/proxy/moveMoneySvc/rtrvPyeeList~~
~~9~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/time/getServerTime~~
~~10~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/time/getServerTime~~
11	200	HTTPS	www.services.online-banking.hsbc.co.uk	/gpib/channel/proxy/utilitiesDataSvc/rtrvReAutnTdsConf
~~12~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/securityProfileService/rtrvTokenDtls?dojo.preventC~~
~~13~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/proxy/moveMoneySvc/rtrvPymtLimit~~
~~14~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/securityProfileService/rtrvTokenDtls?dojo.preventC~~
~~15~~	~~200~~	~~HTTPS~~	~~lo.v.liveperson.net~~	~~/api/js/69833628?sid=-Jk7CfH-TRitbVvPQVZa5Q&cb=lpCb25778x~~
~~16~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/securityProfileService/rtrvTokenDtls?dojo.preventC~~
~~17~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/time/getServerTime~~
18	200	HTTPS	www.services.online-banking.hsbc.co.uk	/gpib/channel/proxy/utilitiesDataSvc/rtrvReAutnTdsConf
~~19~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/time/getServerTime~~
~~20~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/time/getServerTime~~
~~21~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/securityProfileService/rtrvTokenDtls?dojo.preventC~~
~~22~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/time/getServerTime~~
~~23~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/time/getServerTime~~
~~24~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/securityProfileService/rtrvTokenDtls?dojo.preventC~~
25	200	HTTPS	www.services.online-banking.hsbc.co.uk	/gpib/channel/proxy/moveMoneySvc/rtrvBankCde
26	200	HTTPS	www.services.online-banking.hsbc.co.uk	/gpib/channel/proxy/moveMoneySvc/rtrvIntlPymtFxRate
27	200	HTTPS	www.services.online-banking.hsbc.co.uk	/gpib/channel/proxy/moveMoneySvc/prpslNwPyeeIntlPymt
~~28~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/time/getServerTime~~
~~29~~	~~200~~	~~HTTPS~~	~~lo.v.liveperson.net~~	~~/api/js/69833628?sid=-Jk7CfH-TRitbVvPQVZa5Q&cb=lpCb85417x~~
30	409	HTTPS	www.services.online-banking.hsbc.co.uk	/gpib/channel/proxy/moveMoneySvc/confrmNwPyeeIntlPymt
~~31~~	~~200~~	~~HTTPS~~	~~www.services.online-banking.hsbc.co.uk~~	~~/gpib/channel/securityProfileService/verifyCam30Status~~

Figure 6.5: User-bank communication for money transfer to new accounts. Redandunt session are overlined.

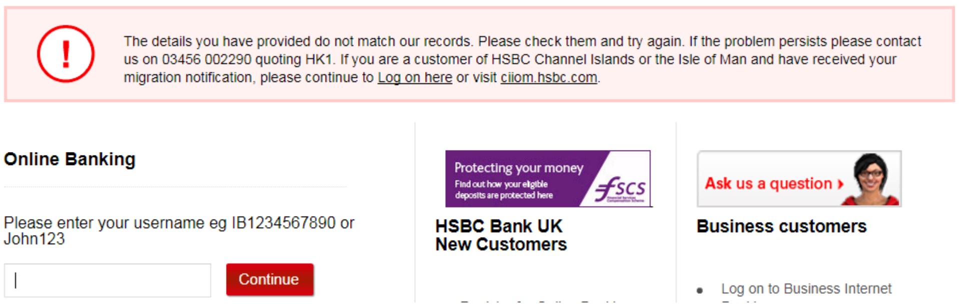
Messing with the transactions and getting personal data from an account are attacks that require a certain set of conditions to be met. Their execution demands the proper planning and expertise, and there is also the possibility of failure. However, in the online banking website lies a flaw far more dangerous that could expose multiple accounts and exploits can be performed by anybody with some basic programming skills.

This flaw is due to the structure of HSBC online banking log on web page, which requires: first the user-name, and then redirects to a new page for providing password and answering security questions. In figure 6.6, we present how the web site looks when a user-name that does not exist is given. The image shows a prompt saying there is no match for the specific user-name. From this, we conclude that if the account does not exist, the ”no match” prompt will appear but if it exists the page is redirected to the password and security questions page. This alone sets every HSBC user-name in danger of exposition since it could verify existing account user-names.

Further more, by getting our account locked and attempting a log-on, we come across the page shown in figure 6.7, whereas a new page notifying that the account is locked, is presented. Further research has shown this happens also for suspended accounts. This is presented in figure 6.8.

Using this user-name exposition, one could attempt log-on with various user-names and find out whether the specific user-name responds to an existing account and if it does, the account state. This information alone could be a huge vulnerab-

,



The details you have provided do not match our records. Please check them and try again. If the problem persists please contact us on 03456 002290 quoting HK1. If you are a customer of HSBC Channel Islands or the Isle of Man and have received your migration notification, please continue to Log on here or visit ciiom.hsbc.com.

Online Banking

Please enter your username eg IB1234567890 or John123

HSBC Bank UK New Customers

Business customers

- Log on to Business Internet

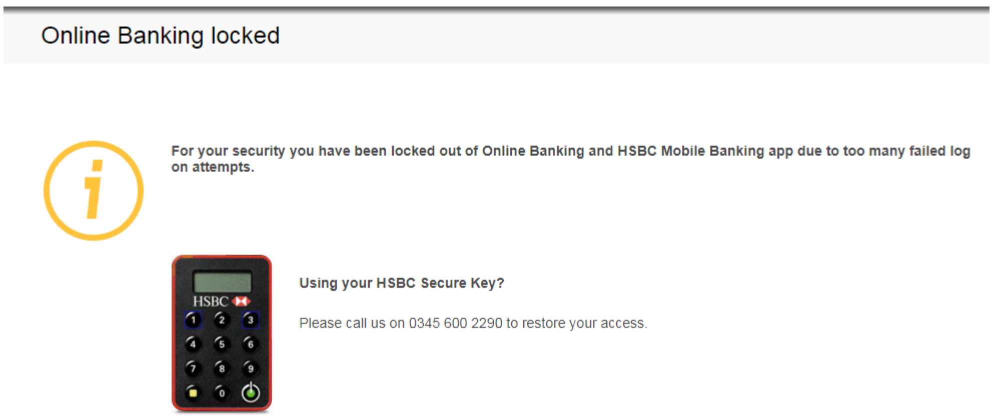
Figure 6.6: Online banking log-on attempt with wrong user name.

ility since the attacker could use the user-name to block any existing account that he finds and perform a denial of service (DoS) attack to the online banking users. This can be achieved by creating a series of failed log-on attempts due to providing incorrect data. Such attack could have the same results as the one happened in 2016 ,when users were locked out for two days [77].

Finally, putting together the data obtained from a MITM attack, the fact that one can know and alter the state of an existing account without the victim noticing, give unlimited options to an attacker with shapr social engineering skills.

An attacker, after achieving a successful MITM and has obtained the victim's data, could attempt to lock the account and even verify that he has succeeded. Then by attempting to unlock the account, he will reach the point where he has to receive the unlock code from bank through email or SMS. The attacker can contact the victim presenting himself as the bank representative and by providing some personal information convince the victim. Then he could notify the victim that their account is locked for some reason, and ask for a reply back with the unlock code that they will be receive in a few minutes. If the attack succeeds the attacker can register his own device on the victim's account and even change log-on password. This way the attack would result into taking complete control of the victim's account.

To conclude, HSBC employs strict state of the art mechanisms to protect its clients, to the point that can be considered "terribly inconvenient" by users as shown in HSBC's mobile banking app review in figure 6.9 However, the way it uses its resources is really poor, and while giving the sensation of a highly secure service it proves to leave users unprotected from being locked out from it to losing their personal data or their money and bank accounts.

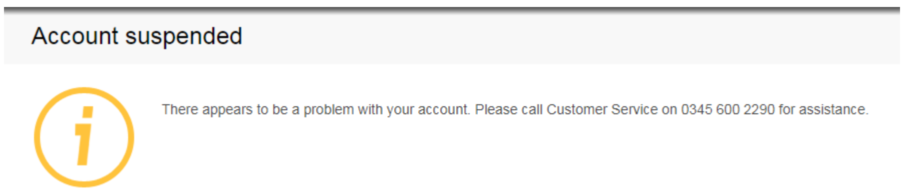

Online Banking locked

For your security you have been locked out of Online Banking and HSBC Mobile Banking app due to too many failed log on attempts.

Using your HSBC Secure Key?

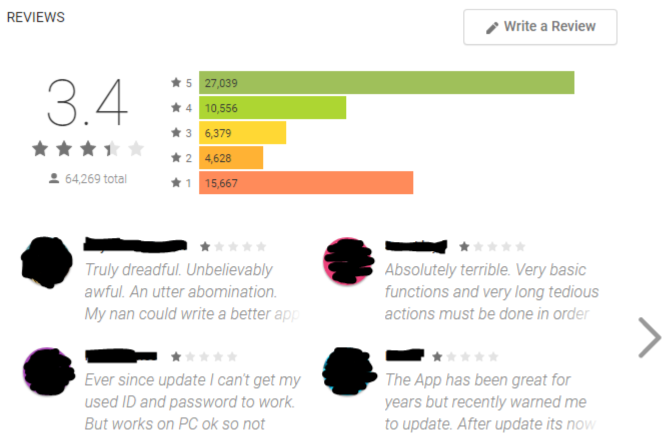
Please call us on 0345 600 2290 to restore your access.

Figure 6.7: Online banking log-on attempt when account is locked.


Account suspended

There appears to be a problem with your account. Please call Customer Service on 0345 600 2290 for assistance.

Figure 6.8: Online banking log-on attempt when account is suspended.


REVIEWS

3.4

64,269 total

★ 5	27,039
★ 4	10,556
★ 3	6,379
★ 2	4,628
★ 1	15,667

[redacted] [redacted] ★
Truly dreadful. Unbelievably awful. An utter abomination. My nan could write a better app

[redacted] [redacted] ★
Absolutely terrible. Very basic functions and very long tedious actions must be done in order

[redacted] [redacted] ★
Ever since update I can't get my used ID and password to work. But works on PC ok so not

[redacted] [redacted] ★
The App has been great for years but recently warned me to update. After update its now

Figure 6.9: HSBC mobile banking app reviews on Google store on 10/7/2017.

Chapter 7

Vulnerability assessment of HSBC OTP generation

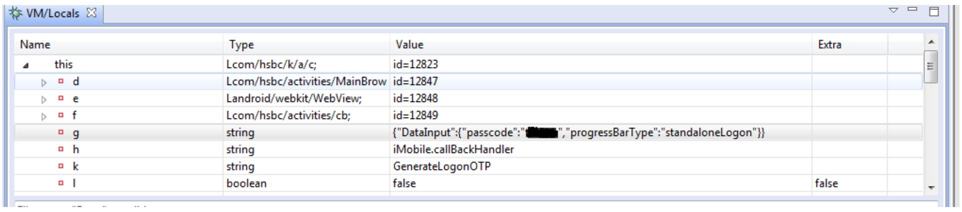
As a final part of our work, we decided to give a look in the insight of OTP generation algorithm and define how secure it is for being replicated. For this part we focus our research on log-on OTP generation process.

7.1 Yet another APK update

Before starting with OTP analysis, we realized that the app had a mandatory update that demanded to be done. Installing the newest version of the APK in a new device, with a quick check up we were able to draw some conclusions about the update:

- APK contains two classes.dex (classes.dex and classes2.dex) instead of just one like the previous version. Classes2.dex seems to contain mostly crypto, x509 certificate and other mathematical methods.
- Using *adb shell* command we are not able to access the HSBC folder getting a ”Permission denied” message.
- Adb backup does not include *Shared_preferences* folder.
- APK not debuggable.

Moving forward, disassembling the APK and using a simple text search for the word ”libshield”, we find that Libshield key word is in 4 different .Smali files. From here we see that there are some changes in the new file since in the previous versions the keyword is found in 293 files. Those files are AppMonitor, LibshieldLoader, LibshieldStarter$ActivityMonitor and LibshieldStarter. However, following the code is clear that AppMonitor.Smali is responsible for calling LibshieldStarter and then the rest are called from there.



Name	Type	Value	Extra
this	Lcom/hsbc/k/a/c;	id=12823	
d	Lcom/hsbc/activities/MainBrow	id=12847	
e	Landroid/webkit/WebView;	id=12848	
f	Lcom/hsbc/activities/cb;	id=12849	
g	string	{"DataInput":{"passcode":"[redacted]","progressBarType":"standaloneLogon"}}	
h	string	iMobile.callBackHandler	
k	string	GenerateLogonOTP	
l	boolean	false	false

Figure 7.1: OTP input password extracted during debugging

To this point, by just removing the two lines of code that call LibshieldStarter and build the app as debuggable we have a fully functional APK that:

- Is debuggable
- Provides access to HSBC folder through adb shell
- Backup returns shared preferences folder

Finally, we transferred the registration to the new device and started with OTP analysis.

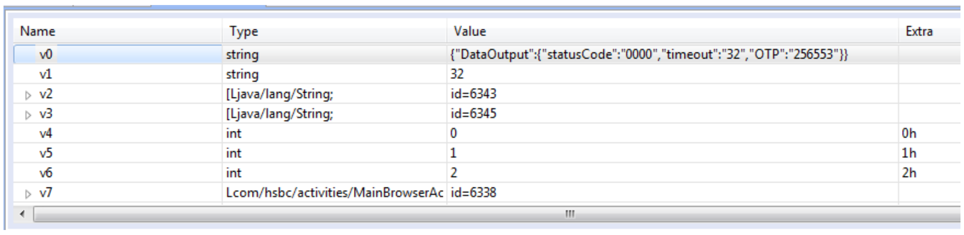
7.2 Locating OTP generator

After successfully transferring the registration to the new device, at first we created a backup of the APK. This time backup included shared preferences folder and specifically a file called *SOTPValueStore_148*, which contains static vector (SV), dynamic vector (DV), encryptedGUID, clientServerTimeShift and soft_token_random _number.

Keeping that in mind, a followup search is conducted with SOTPValueStore as the text key. The search revealed the file com/hsbc/k/c/g.Smali that seemed to be involved in fetching the data from SOTPValueStore. Following the key search, with the key word ”generate”, lead to com/hsbc/k/a/c.Smali and com/hsbc/util/a/ag.Smali that are both calling a function with ”GenerateLogonOTP” string as a parameter.

At this point we used aused a trial version of an Android debugging tool, JEB [81], that gives the flexibility to debug .Smali code while apk is running. Setting break points to the files found in the previous step we were ready to start the debugging process.

During debugging we were able to see the input password, input parameters and OTP code in the system registers as shown in figure 7.1 and figure 7.2. At this point we were able to identify the entry points to OTP generation algorithm.



Name	Type	Value	Extra
v0	string	{"DataOutput":{"statusCode":"0000","timeout":"32","OTP":"256553"}}	
v1	string	32	
v2	[Ljava/lang/String;	id=6343	
v3	[Ljava/lang/String;	id=6345	
v4	int	0	0h
v5	int	1	1h
v6	int	2	2h
v7	Lcom/hsbc/activities/MainBrowserAc	id=6338	

Figure 7.2: OTP code extracted during debugging

7.3 OTP function calls

Using dynamic analysis we managed to reach the entry points of the OTP algorithm. Specifically we consider the ”root entry point” to be in the file com/hsbc/k/c/a.Smali.

However, to move forward and be able to understand what happens in each function we decided to perform a static analysis of the code. In order to achieve that the .Smali code is decompiled to java to make it easier to read.

Starting from our entry point in com/hsbc/k/c/a.java we realized that the method

```
public static com.a.a.a.a.f a(Context context, String
    Password, int log\_int\_1, boolean bl2, String[]
    accnt\_ID)
```

is responsible for receiving SV, DV from ValueStore_148, perform some checks on them and then do a Base64 decode on each vector. After that it will retrieve clientServerTimeShift from SOTPValueStore_148 and based on the input parameter bl2 will call proceed for transaction OTP generation or logon and registration OTP generation. The calling code is the following:

```
 if (bl2) {
         return com.a.a.a.a.a(SV, DV, accnt_ID,
             Password, 12, log_int_1, null);
     }
     return com.a.a.a.a.a(SV, DV, Password, 12,
         log_int_1, null);
```

At this point it should be noted that input parameters and main variables were renamed in order for an easier understanding. Specifically, the inputs to our root entry point function are explained in table 7.1 and those names are going to be used in future references.

Following the code we move to a/a/a/a.java where a method is called with sole purpose to add some constant boolean and null values to the given parameter set and call a new method from ai.java. The called method is considered to be the OTP generator main function responsible for controlling the whole OTP generation process. The function is presented in the code below:

Context context	Related to SOTPValueStore_148 directory
String Password	OTP input password
int log_int_1	1− > logon, 2− > reauth, 3− > transaction
boolean bl2	1 if transaction OTP, 0 for reauth or login
String[] accnt_ID	transfer account digits

Table 7.1: Inputs variables for OTP root entry point.

```
public static com.a.a.a.a.f a(byte[] SV, byte[] DV,
   long clientServerTimeShift, int log_int_1, String
   Password,  byte[] arrby_null, boolean bl2_false,
   String string3_null, boolean bl3_false, String[]
   arrstring_null, boolean bl4_null, String
   string4_null, String string5_null, boolean bl5_null
   ) {
    block4: {
        object_c = ai.a((byte[])SV, (byte[])DV,
           log_int_1, Password, arrby_null, bl2_false,
            string3_null, bl3_false, arrstring_null,
           bl4_null, string5_null, bl5_null);
        if (object_c.b() == 0)
            break block4;
        Object_com_a_a_a_a_f = new com.a.a.a.a.f(
           object_c.b(), object_c.a.c.b, l.a(object_c.
           a), object_c.c, null, null);
        object_c.a.a();
        return Object_com_a_a_a_a_f;
    }
    try {
        Object_com_a_a_a_a_f = ai.a(object_c.a,
           log_int_1, object_c.b, string4_null,
           clientServerTimeShift);
        Object_c.a.a();
        return Object_com_a_a_a_a_f;
    }
    catch (j j2) {
        return ai.d(j2.a());
    }
    catch (Exception exception) {
        n.a("Unknown error", exception);
        return ai.d(60537);
    }
}
```

At first, some functions are performed using SV, DV, password, and and log_int_1 and an object of c.class is generated. Then using the new object a second object from com.a.a.a.a.f.class is generated by feeding the parameters of the first object, clientServerTimeShift and log_int_1 value. The second object contains the new DV and OTP. Finally, if the outputs pass a series of tests and no exceptions are thrown during the process the object is returned from the function, DV is stored in memory and OTP is presented to the user.

Further work is done to follow up the function calls in order to gain a deeper understanding of OTP working, and the level of obfuscation employed. However, before diving into the OTP algorithm, we use the following paragraphs to present some of the the protection mechanisms that the app employs in order to make code understanding more challenging.

7.4 HSBC code protection mechanisms

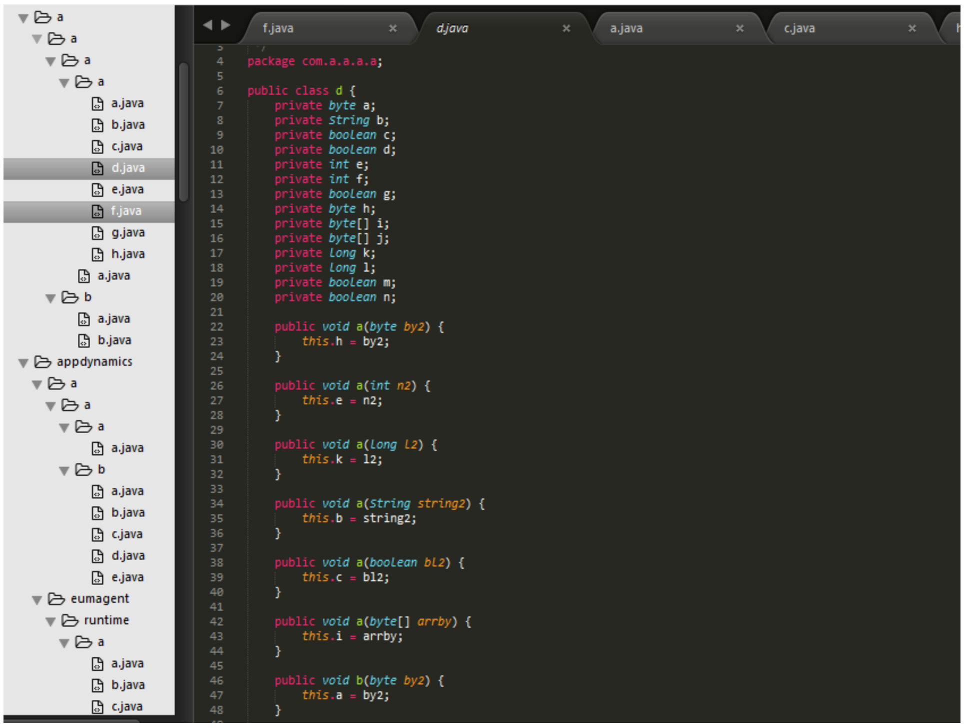
In the previous sections we discussed about HSBC's security mechanisms on online banking and seen some security holes that could permit an attacker to redirect money or steal personal information from the user. However, those vulnerabilities need to meet certain conditions to be enabled and also depend on how, when and where the app is used. Besides that, if a malicious person is able to set hands on OTP generation algorithm and OTP related keys which are stored in app memory the results could be tragic for the victim.

The attacker could be able to reproduce OTP codes and even take over the bank account. Based on that, we consider OTP generation algorithm to be one of HSBC's most valued algorithms when it comes to client service. This statement is also supported by the obfuscation mechanisms that the application employees in order to hide their algorithm, data structures and logic from being being revealed to unauthorized entities.

According to Collberg et al [82] obfuscation is a process of transforming a source program into a target program that behaves the same way but is more complex to follow and understand. In this section we are identifying some of HSBC's obfuscation techniques that are used to make the code understanding a time consuming task.

7.4.1 Renaming Identifiers

In general HSBC renames paths, class names, methods and variables using the alphabet. As shown in figure 7.3 and in previous code samples, names like "a/a/a/f" are quite common and considering the large number of identifiers with similar names ("a/a/a/a.class" and "a/a/a.class") following through becomes confusing. The replacement of those identifiers cannot be undone but it's not impossible to understand the code either, since it can be approached as a code with poor choice of variable names.



```java
package com.a.a.a.a;

public class d {
    private byte a;
    private String b;
    private boolean c;
    private boolean d;
    private int e;
    private int f;
    private boolean g;
    private byte h;
    private byte[] i;
    private byte[] j;
    private Long k;
    private Long l;
    private boolean m;
    private boolean n;

    public void a(byte by2) {
        this.h = by2;
    }

    public void a(int n2) {
        this.e = n2;
    }

    public void a(Long L2) {
        this.k = l2;
    }

    public void a(String string2) {
        this.b = string2;
    }

    public void a(boolean bl2) {
        this.c = bl2;
    }

    public void a(byte[] arrby) {
        this.i = arrby;
    }

    public void b(byte by2) {
        this.a = by2;
    }
```

Figure 7.3: Example of renaming identifiers in decompiled HSBC code.

7.4.2 Call Indirection

Call Indirection refers to the case where having a call to some method A is altered to a new method B that will then call method A. This technique is used in the very first steps on the root entry point of OTP algorithm, when com.a.a.a.a.a method is called with sole purpose to call a method from ai.java. The decompiled code samples of this example are shown in figure 7.4.

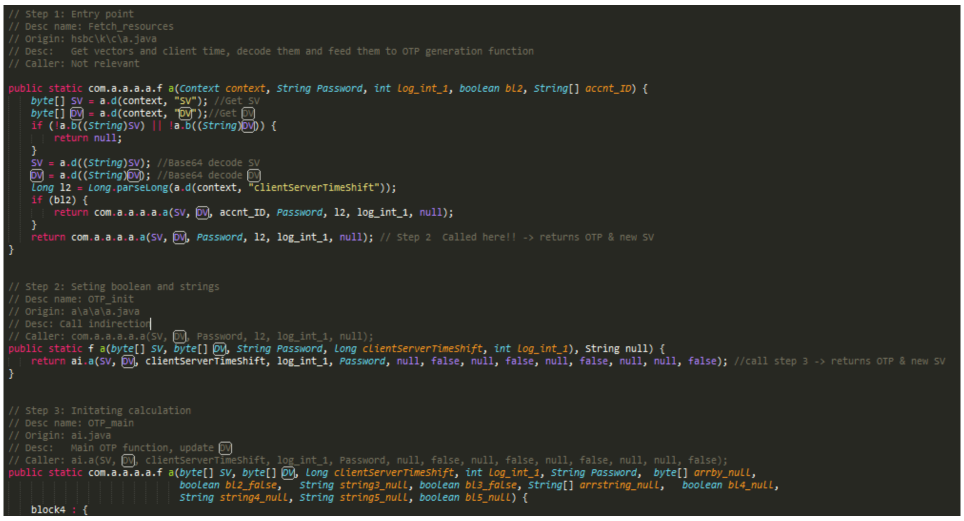
7.4.3 Code reordering

Code reordering refers to altering the order of instructions in the program. In HSBC's code this is shown through the numerous *GOTO* commands in the Smali and decompiled code. An example of code reordering in the decompiled code is presented in figure 7.5

7.4.4 Arithmetic obfuscation

Making an arithmetic function more complex than it should be. In the following code we present a simple example of such technique found on the disassembled code:

```
.method public static a(JJ)Z
```



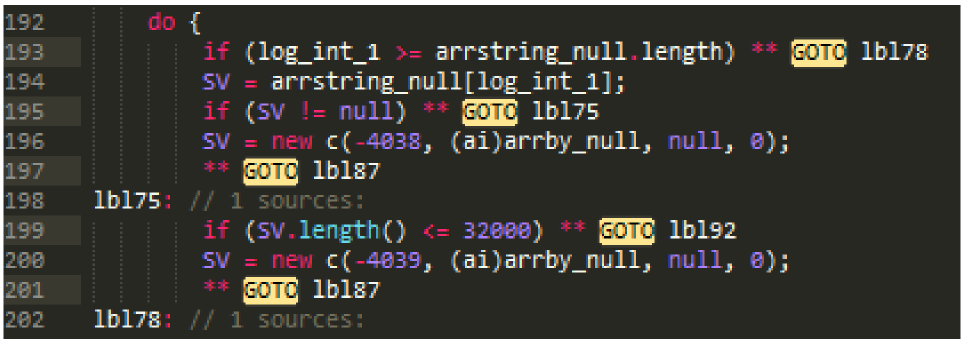
```java
// Step 1: Entry point
// Desc name: Fetch_resources
// Origin: hsbc\k\c\a.java
// Desc:   Get vectors and client time, decode them and feed them to OTP generation function
// Caller: Not relevant

public static com.a.a.a.a.f a(Context context, String Password, int log_int_1, boolean bl2, String[] accnt_ID) {
    byte[] SV = a.d(context, "SV"); //Get SV
    byte[] DV = a.d(context, "DV");//Get DV
    if (!a.b((String)SV) || !a.b((String)DV)) {
        return null;
    }
    SV = a.d((String)SV); //Base64 decode SV
    DV = a.d((String)DV); //Base64 decode DV
    long l2 = Long.parseLong(a.d(context, "clientServerTimeShift"));
    if (bl2) {
        return com.a.a.a.a.a(SV, DV, accnt_ID, Password, l2, log_int_1, null);
    }
    return com.a.a.a.a.a(SV, DV, Password, l2, log_int_1, null); // Step 2  Called here!! -> returns OTP & new SV
}


// Step 2: Seting boolean and strings
// Desc name: OTP_init
// Origin: a\a\a\a.java
// Desc: Call indirection
// Caller: com.a.a.a.a.a(SV, DV, Password, l2, log_int_1, null);
public static f a(byte[] SV, byte[] DV, String Password, Long clientServerTimeShift, int log_int_1), String null) {
    return ai.a(SV, DV, clientServerTimeShift, log_int_1, Password, null, false, null, false, null, false, null, null, false); //call step 3 -> returns OTP & new SV
}


// Step 3: Initating calculation
// Desc name: OTP_main
// Origin: ai.java
// Desc:   Main OTP function, update DV
// Caller: ai.a(SV, DV, clientServerTimeShift, log_int_1, Password, null, false, null, false, null, false, null, null, false);
public static com.a.a.a.a.f a(byte[] SV, byte[] DV, Long clientServerTimeShift, int log_int_1, String Password,  byte[] arrby_null,
                              boolean bl2_false,   String string3_null, boolean bl3_false, String[] arrstring_null,   boolean bl4_null,
                              String string4_null, String string5_null, boolean bl5_null) {
    block4 : {
```

Figure 7.4: Example of call indirection in decompiled HSBC code.



```java
        do {
            if (log_int_1 >= arrstring_null.length) ** GOTO lbl78
            SV = arrstring_null[log_int_1];
            if (SV != null) ** GOTO lbl175
            SV = new c(-4038, (ai)arrby_null, null, 0);
            ** GOTO lbl87
lbl175: // 1 sources:
            if (SV.length() <= 32000) ** GOTO lbl92
            SV = new c(-4039, (ai)arrby_null, null, 0);
            ** GOTO lbl87
lbl78: // 1 sources:
```

Figure 7.5: Example of code reordering in decompiled HSBC code with multiple GOTO commands.

```
    .locals 2

    and-long v0, p0, p2

    cmp-long v0, v0, p2

    if-nez v0, :cond_0

    const/4 v0, 0x1

    :goto_0
    return v0

    :cond_0
    const/4 v0, 0x0

    goto :goto_0
.end method
```

where the commands *and-long v0, p0, p2* and *cmp-long v0, v0, p2* could just be *cmp-long v0, p0, p2* making the first command redundant.

7.4.5 Really complex call graph

Observing the first 30 method calls in call graph starting with OTP entry point, we find out that HSBC employs a really long call graph with more than 20 function calls depth that are called repeatedly over and over again.

7.5 OTP main algorithm

Having discussed on the importance of OTP and the protection mechanisms in the code is time to give a look at the actual algorithm. Due to legal issues we present a high level approach to the algorithm instead of a detailed explanation without revealing any static codes, keys etc. Due to the obfuscation employed in the app it would be really hard to explain all the different functions and results thus we group them together in simpler steps and provide our own, easy to understand names for inputs, outputs and functions involved.

For the purposes of our work we analyze two OTP methods out of the three, log-on OTP and transaction OTP. Their process is quite similar with some minor differences. At first, both algorithms need to do a pre-processing to extract codeA from SV and vector second, logonCounter and codeB from DV. Combining codeA, codeB, password and a static code, hard-coded in the app (static1) a 32 byte code is obtained. Passing that bytecode through two SHA256 hashing functions and some small byte processing in between (SHAfunc1), vector first is generated. Vector first is fed to a function (keyGen1) in order to generate a 10 by 4 matrix used as an AES key, key1. Finally, vector second is processed (IVgen) in order to extract the AES initial vector (IV) and using key1, a 10 round AES encryption is performed. The result of the AES encryption is processed in order to come to the form of a 16 byte vector, vector third.

For log-on OTP, vector third is fed to a function (keyGen2) and produces a new 10 by 4 matrix, the logonKey. KeyGen2 is identical to keyGen1 with one step less. At this point dynamic data are used. First the logonCounter is read and a timestamp is generated. Those two are processed (timeFunc) to generate the ts vector and then the logonCounter is increased by 1. In the next step ts vector is passed through IVgen in order to output the IV for the next AES encryption. After passing IV and logonKey through a 10 round AES encryption algorithm the result is broken to a 16 byte vector. The bytes are then mixed, processed and tured to a number vector, numVec using the mixing function mixer. Finally the 2nd number will be replaced by $logonCounter \% 10$ and the first 6 numbers are outputted as the OTP.

Transaction OTP generation involves some extra steps compared to logon OTP. After vector third generation, third is combined with codeA, codeB and another hard-coded value (static2) to generate a new 32 byte code. The generated code is fed to a SHAfunc2, a very similar process to SHAfunc1 that also involves two SHA256 hashing functions. This results to vector fourth generation that when fed to keyGen2 will output a 10 by 4 matrix, the txnKey. Similar to OTP transactionCounter and a generated timestamp are combined to provide the timestamp vector. However, this time timestamp vector also includes the last 4 digits of the transaction account. After timestamp generation the process is identical to log-on OTP with the use of transactionCounter instead of logonCounter in the final step.

The OTP algorithm for both transaction and logon is shown in figure 7.6, using the function and vector names explained in the previous paragraphs.

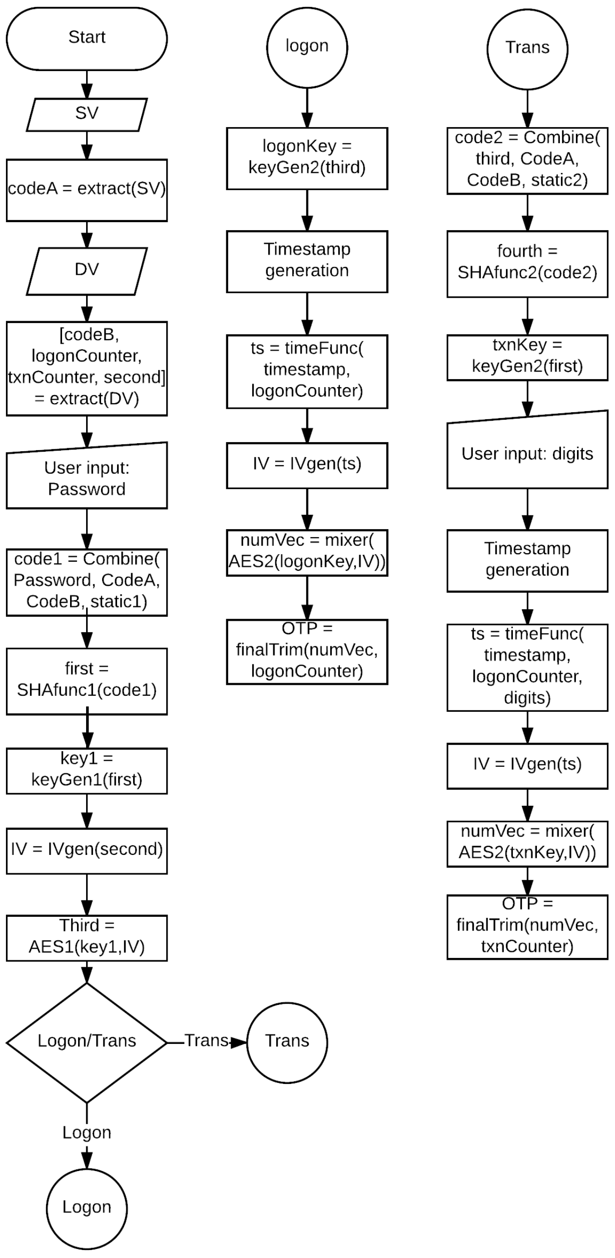

Start
SV
codeA = extract(SV)
DV
[codeB, logonCounter, txnCounter, second] = extract(DV)
User input: Password
code1 = Combine(Password, CodeA, CodeB, static1)
first = SHAfunc1(code1)
key1 = keyGen1(first)
IV = IVgen(second)
Third = AES1(key1,IV)
Logon/Trans
Trans
Trans
Logon
Logon
logon
logonKey = keyGen2(third)
Timestamp generation
ts = timeFunc(timestamp, logonCounter)
IV = IVgen(ts)
numVec = mixer(AES2(logonKey,IV))
OTP = finalTrim(numVec, logonCounter)
Trans
code2 = Combine(third, CodeA, CodeB, static2)
fourth = SHAfunc2(code2)
txnKey = keyGen2(first)
User input: digits
Timestamp generation
ts = timeFunc(timestamp, logonCounter, digits)
IV = IVgen(ts)
numVec = mixer(AES2(txnKey,IV))
OTP = finalTrim(numVec, txnCounter)

Figure 7.6: OTP generation algorithm flow chart for log-on and transaction OTPs.

Chapter 8

HSBC open-source banking

We now present the start of an HSBC application with superior security. Our work presents some fixes to our discovered vulnerabilities. The work presented in previous chapters enables us to create all core banking operations in a fully transparent and open source methodology. We believe this open ecosystem approach will yield superior security and faster response to discovery of new vulnerabilities.

Disclaimer, this work is only partially finished. Creating a full features stable banking app is not a single person project, but we successfully started. In our application we included the functions discussed in section 6:

- Mobile banking: log-in using Password
- Mobile banking: log-in using OTP
- Mobile banking: balance view
- Online banking: log-in using OTP
- Online banking: balance view
- Online banking: transaction history
- Online banking: view known accounts
- Online banking: transaction to known European accounts
- Online banking: transaction to new European accounts
- OTP: OTP generation for log on
- OTP: OTP generation for transactions

In this section we provide a description of our HSBC open-source banking application and its features.

8.1 Application and code size

HSBC open-source banking app is a command line interface were the user can perform various banking functions around HSBC mobile and online banking infrastructure. It is implemented using python and requests [79] library and its main function is to reproduce the communication sessions that the other HSBC banking apps use in a more efficient way, removing redundant sessions and unnecessary information exchange. The implementation consists of two parts, the first one is the HSBC_manager class that is responsible for client-bank connection, performing the communication sessions and providing the basic communication with the user. The second part is a custom made library HSBC_requests that includes the JSON templates for each session, processes the banks responses to get the desired information through web scraping functions using beautiful soup library or custom made functions for JSON reading and finally, it provides user dialog functions. In addition to banking related code, an extra library: OTP, handles OTP generation for log on and transaction.

HSBC_manager is implemented within 340 lines of python code and HSBC_requests within 845 (both including comments and empty lines). The size for HSBC_manager is 19 KB and HSBC_requests is 33 KB. OTP library is implemented in 880 lines of code and total size of 40KB. Compared to the original 483 KB zipped APK code of the mobile banking app and the 27.7 MB of the whole HSBC app, our open-source API is almost 5 times smaller in size and can perform transactions to both known and new accounts.

8.2 Function calls

In this section we provide a more detailed description of the main class HSBC_manager functions is presented as well as the main workings of HSBC_requests library. Finally the system flow is presented and explained.

8.2.1 HSBC_manager

HSBC_manager contains the six main functions of the class which are initialization three different log-in methods and the two different kinds of transactions. It is responsible for completing the necessary communication sessions and calling the suitable functions from HSBC_requests library to process the responses and print user dialogues.

__init__(): Initial function for creating the class. At this point, the session objects are created and certificate verification is decided by hard-coding variable debug. When debug is set to 0, then a secure connection is established while otherwise the communication is forwarded to fiddler.

log_in_AppOTP(): This function is responsible for performing a mobile banking app log-in using OTP. The log-in happens in seven communication sessions responsible for are SAAS authentication token fetch, SAAS authentication, provid-

ing credentials, verifying authentication and retrieving account information. Finally it prints account balances.
log_in_AppPassword: Similar to log_in_AppOTP(), this function performs a mobile banking app log-in using password. The log-in happens in nine communication sessions and in the end, it prints account balances.
log_in_WebOTP: This function performs an online banking log-in within eight sessions starting with two GET requests, authentication using userID, memorable answer and OTP provided by the user, authentication verification and finishes with two sessions, one to retrieve account information and extract account_index and another to receive transaction history and print it through TransactionSummary(resp) function. Finally, this function provides a dialogue for user to chose between new transaction, known transaction or exit the application.
At this point should be noted that for automating the log-in task, userID, memorable answer and password are hard-coded while OTP should be provided through a registered device. This way security is preserved since without the trusted device the OTP cannot be obtained.
known_transaction(Header_Services,Account_index): Using this function, user can perform transactions to known accounts as mentioned in section 6. This function is completed within seven sessions where the user has to chose the account to send money to, amount and date and the rest is done automatically.
new_transaction(Header_Services,Account_index): This function is responsible for performing transactions to new accounts as described in section 6. The user provides as input the transaction amount, IBAN, country code, name, address, city and date of transaction and the rest is completed automatically in eight sessions.

8.2.2 HSBC_requests

HSBC_requests library contains:

- Five different static header templates for mobile banking log-in
- Five different static header templates for online banking log-in
- One dynamic header template for online banking transactions
- Four static JSON templates for mobile banking log-in
- Five dynamic JSON generation functions for mobile banking log-in
- Four static JSON templates for online banking log-in
- Eight dynamic JSON generation functions for online banking log-in and transactions
- Seven user dialogue functions for presenting account data or receiving user input

The functions included in HSBC_requests are the following:
showAccounts(resp): Gets a session response and prints account information and balance after mobile banking log-in.
AccountSummary(resp): Gets a session response and prints account information and balance after online banking log-in. Also returns account_index.
TransactionSummary(resp): Gets a session response and prints transaction history after online banking log-in.
get_payee(resp): Gets a session response and prints the list of known accounts after user chooses to perform transaction to known account. Then asks user to choose an account and returns the account information.
get_account(resp): Gets a session response and returns account_index without any printing.
get_txn_details(): User dialog for getting the necessary data for a transaction to a known account.
get_new_txn_details(): User dialog for getting the necessary data for a transaction to a new account.

Modes of operation - certificate verification

As mentioned in the beginning of this section, an option of SSL certification is added to our application. When the application is used for debugging, packet capture is needed thus SSL certification can be disable and when it is used as a banking service SSL certification should be enabled.

When certificate verification is used, the app introduces a layer of protection to prevent from MITM attacks. In order to achieve that Mozila trust store is used which is included in python's requests library [79]. It contains the certificates of root certificate authorities of witch HSBC is part of their certificate chain and expires in 2036 [80].

Chapter 9

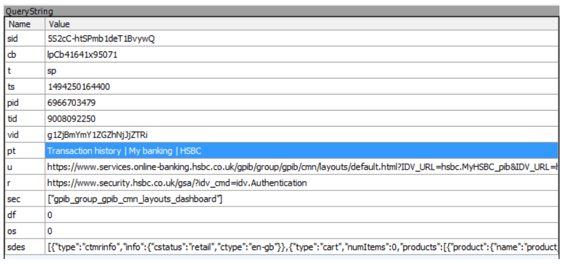
Permisionless banking API evaluation

In the previous sections we introduced the idea of an open banking API, presented the inner workings of a mobile banking system and implemented our own open version of it. For the following paragraphs we are going to evaluate our API comparing it to the original mobile and online banking app in three different aspects:

- Un-necessary data
- Performance evaluation
- Flexibility

9.1 Un-necessary data

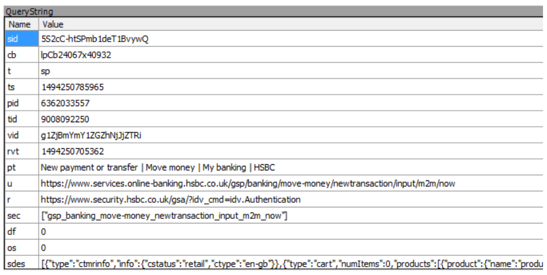
According to [83], after analyzing 500 mobile apps, MIT researchers found out that almost 50 percent of communication channels do not relate with user experience. In some cases those channels are associated with with other companies' servers. By checking the traffic from online banking and iOS mobile banking app, we found out that both platforms send packets to other hosts besides HSBC. Specifically, online banking sends information to *lo.v.liveperson.net*, a company with products including in-app messaging, real-time analytics and customer care bots [84]. In figure 9.1, we present two screen shots from the WebForm data sent to this host. Data from image (a) are sent during web log-on where, transaction history is presented to the user within the first screen. In image (a) there is a field with value *Transaction history — My banking — HSBC* which seems to keep track on what the user is going to see or chose with log-on. Further more, image (b) shows a WebForm captured while attempting to make a new transaction, and similar to image (a) the field value contains the information *New payment or transfer — Move money — My banking — HSBC*.



QueryString

Name	Value
sid	5S2cC-htSPmb1deT1BvywQ
cb	lpCb41641x95071
t	sp
ts	1494250164400
pid	6966703479
tid	9008092250
vid	g1ZjBmYmY1ZGZhNjJjZTRi
pt	Transaction history | My banking | HSBC
u	https://www.services.online-banking.hsbc.co.uk/gpib/group/gpib/cmn/layouts/default.html?IDV_URL=hsbc.MyHSBC_pib&IDV_URL=
r	https://www.security.hsbc.co.uk/gsa/?idv_cmd=idv.Authentication
sec	["gpib_group_gpib_cmn_layouts_dashboard"]
df	0
os	0
sdes	[{"type":"ctmrinfo","info":{"cstatus":"retail","ctype":"en-gb"}},{"type":"cart","numItems":0,"products":[{"product":{"name":"product

(a) Data sent while log-on



QueryString

Name	Value
sid	5S2cC-htSPmb1deT1BvywQ
cb	lpCb24067x40932
t	sp
ts	1494250785965
pid	6362033557
tid	9008092250
vid	g1ZjBmYmY1ZGZhNjJjZTRi
rvt	1494250705362
pt	New payment or transfer | Move money | My banking | HSBC
u	https://www.services.online-banking.hsbc.co.uk/gsp/banking/move-money/newtransaction/input/m2m/now
r	https://www.security.hsbc.co.uk/gsa/?idv_cmd=idv.Authentication
sec	["gsp_banking_move-money_newtransaction_input_m2m_now"]
df	0
os	0
sdes	[{"type":"ctmrinfo","info":{"cstatus":"retail","ctype":"en-gb"}},{"type":"cart","numItems":0,"products":[{"product":{"name":"produ

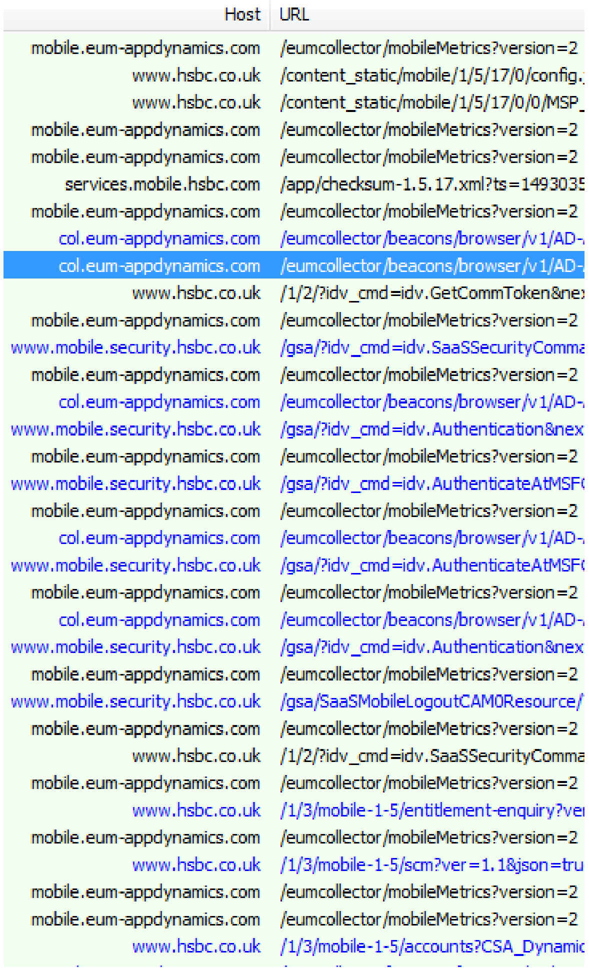
(b) Data sent when attempting a transaction

Figure 9.1: Data sent to lo.v.liveperson.net while using online banking.

Similar to that, on iOS mobile banking app there is a communication with a host other than HSBC. This time the hosts are *mobile.eum-appdynamics.com* and *col.eum-appdynamics.com*, both from appdynamics, an analytics platform which describes it services as ”Monitor every critical swipe, tap, and click” [85]. Regarding the frequency of sessions between app and appdynamics, during a log-on, around 2/3 of the total sessions are associated with appdynamics. A sample of the log-on session is shown in figure 9.2. After an attempt to read the session content, we see that *col.eum-appdynamics.com* sends a JSON form with information regarding the log-on and timestamp, while *mobile.eum-appdynamics.com* seems to sent data in an encrypted format. Deciphering the content is not in the purposes of this work thus the actual data sent to this third party host are unknown.

Coming back to our work, while implementing our own online and mobile banking version, we did not include any third party hosts besides HSBC related. That choice is made due to the reason that packets sent to third parties do not return any useful data, other than empty acknowledgments and banking functions are able to be performed without them. Adding to that, our fully functional code proves that those data are not needed for actual banking functions making them redundant for the user, and consume extra data and time resources while they have no clear purpose.

Sending redundant data when using the online and mobile banking does not apply only when data is sent to third parties, but also within the bank sessions. Redundant sessions between the bank-user communication can be considered: Sessions to get data that the user did not ask to see and the ones that do not affect the process that user selected (log-on, view balance, transaction etc.).



Host	URL
mobile.eum-appdynamics.com	/eumcollector/mobileMetrics?version=2
www.hsbc.co.uk	/content_static/mobile/1/5/17/0/config.
www.hsbc.co.uk	/content_static/mobile/1/5/17/0/0/MSP_
mobile.eum-appdynamics.com	/eumcollector/mobileMetrics?version=2
mobile.eum-appdynamics.com	/eumcollector/mobileMetrics?version=2
services.mobile.hsbc.com	/app/checksum-1.5.17.xml?ts=1493035
mobile.eum-appdynamics.com	/eumcollector/mobileMetrics?version=2
col.eum-appdynamics.com	/eumcollector/beacons/browser/v1/AD-
col.eum-appdynamics.com	/eumcollector/beacons/browser/v1/AD-
www.hsbc.co.uk	/1/2/?idv_cmd=idv.GetCommToken&ne
mobile.eum-appdynamics.com	/eumcollector/mobileMetrics?version=2
www.mobile.security.hsbc.co.uk	/gsa/?idv_cmd=idv.SaaSSecurityComm
mobile.eum-appdynamics.com	/eumcollector/mobileMetrics?version=2
col.eum-appdynamics.com	/eumcollector/beacons/browser/v1/AD-
www.mobile.security.hsbc.co.uk	/gsa/?idv_cmd=idv.Authentication&nex
mobile.eum-appdynamics.com	/eumcollector/mobileMetrics?version=2
www.mobile.security.hsbc.co.uk	/gsa/?idv_cmd=idv.AuthenticateAtMSF
mobile.eum-appdynamics.com	/eumcollector/mobileMetrics?version=2
col.eum-appdynamics.com	/eumcollector/beacons/browser/v1/AD-
www.mobile.security.hsbc.co.uk	/gsa/?idv_cmd=idv.AuthenticateAtMSF
mobile.eum-appdynamics.com	/eumcollector/mobileMetrics?version=2
col.eum-appdynamics.com	/eumcollector/beacons/browser/v1/AD-
www.mobile.security.hsbc.co.uk	/gsa/?idv_cmd=idv.Authentication&nex
mobile.eum-appdynamics.com	/eumcollector/mobileMetrics?version=2
www.mobile.security.hsbc.co.uk	/gsa/SaaSMobileLogoutCAM0Resource/
mobile.eum-appdynamics.com	/eumcollector/mobileMetrics?version=2
www.hsbc.co.uk	/1/2/?idv_cmd=idv.SaaSSecurityComm
mobile.eum-appdynamics.com	/eumcollector/mobileMetrics?version=2
www.hsbc.co.uk	/1/3/mobile-1-5/entitlement-enquiry?ve
mobile.eum-appdynamics.com	/eumcollector/mobileMetrics?version=2
www.hsbc.co.uk	/1/3/mobile-1-5/scm?ver=1.1&json=tru
mobile.eum-appdynamics.com	/eumcollector/mobileMetrics?version=2
mobile.eum-appdynamics.com	/eumcollector/mobileMetrics?version=2
www.hsbc.co.uk	/1/3/mobile-1-5/accounts?CSA_Dynami

Figure 9.2: Packets exchanged during a log-on using iOS mobile banking app. We see that sessions related to eum-appdynamics host appear more often than sessions related to hsbc host.

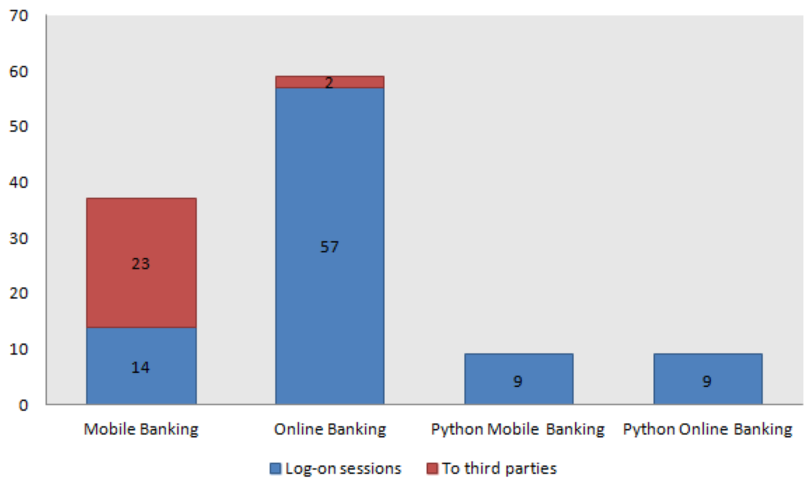


Figure 9.3: Number of sessions involved in log-on process for mobile and online banking.

While developing our own mobile and online banking versions, we attempt to remove redundant sessions and keep only the ones, vital for the task that the user requests. Doing this, we found out that for both online and mobile banking, a lot of unnecessary sessions were involved. As a comparison, we break down the number of sessions that take place during a log-on, for online/mobile and our banking platform and categorize them into *Log-on sessions*, and *To third parties*. In our analysis we did not count sessions related with user interface, including images or javascript code since our application does not use any. Finally, we present our findings in figure 9.3. From the graphs we see that mobile banking app sessions are mostly between third parties leaving 1/3 of the whole communication to take place between the actual bank service. On the other hand, online banking involves less communication with third parties, but sends a huge amount of redundant data, including three sessions filled with multiple ”lorem ipsum” paragraphs.

In another test, we measure the number of bytes sent and received during log-on for online, mobile banking and our python online banking version. The results are presented in figure 9.4. Measuring the amount of data sent and received, reveals that for mobile banking, sessions to third parties are short and contain a small size of data. For online banking, the opposite behaviour is observed. Even though during log-on only two sessions take place outside of HSBC host, the size of data received is quite large.

Finally, similar measurements are done for transaction to a new payee between HSBC online banking and our online banking app. The results regarding number of sessions involved are shown in figure 9.5. In figure 9.6 we present the amount of bytes sent and received during each process.

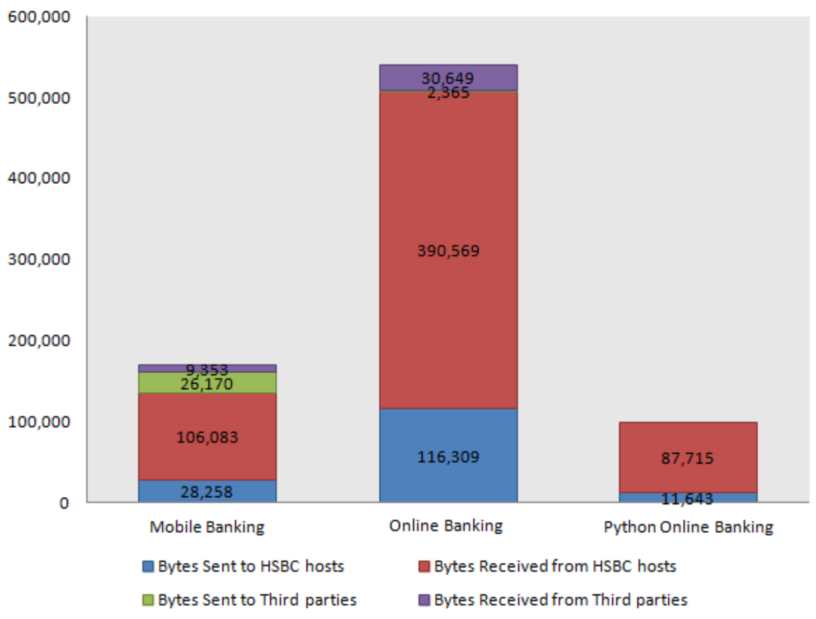


Figure 9.4: Bytes sent/received during log-on process for mobile and online banking.

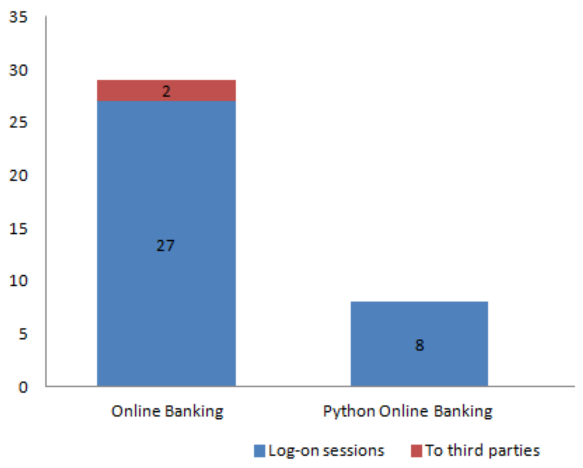


Figure 9.5: Number of sessions involved in a new transaction for HSBC's and our version of online banking.

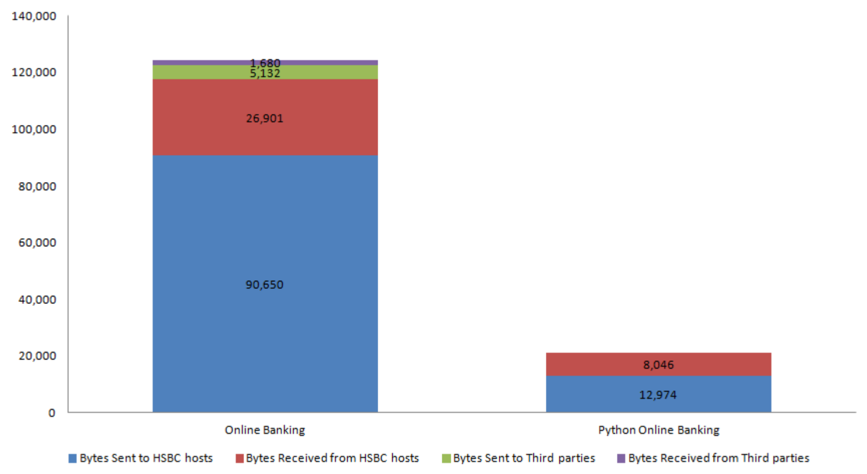


Figure 9.6: Bytes sent/received during a new transaction for HSBC's and our version of online banking.

9.2 Performance evaluation

Considering the redundant amount of bytes sent and received within mobile and online banking apps, in this discussion we evaluate the speed performance of our application compared to the bank provided platforms. To compare the speed, we measure the aggregate session duration for log-on and a successful transaction. Aggregate session duration is the sum of the duration from all involved sessions, timed at the start of the request to the end of the response. This way we have a clear time measurement without involving random delays due to user interactions. The results are presented in table 9.1.

	Mobile Banking	Online Banking	Python online banking
Log-on Duration	03.133 ms	09.973 ms	01.612 ms
Transaction Duration	N.A.	07.647 ms	04.853 ms

Table 9.1: Aggregate session duration for log-on and transaction process.

In another attempt to evaluate our OTP generator, we measure the time that is required for a log-on OTP to be generated by a registered device and our API. Our application generated the OTP in around 1.23 sec, while the registered device took 3.82 sec. This means our application is almost 3 times faster than the registered device when comes to generate a log-on OTP.

9.3 Flexibility

The final aspect is more of a discussion than testing and measurements. From HSBC, we have the HSBC mobile banking that is restricted on transactions and has some really bad reviews from the users while online banking provides more to the user while lacking the security and always needs a trusted device to operate fully. In addition, communication between application and bank includes redundant and questionable exchange of data, something that the average user might not be aware of.

Introducing an open banking API, where everybody could have access to the source code gives the users the privilege to know exactly what happens behind the scenes and customize the app by adding new features or a user interface of their liking. For the purposes of our work we limited the banking functions into log-on, transactions, OTP generation and viewing balances and history, which we consider the most important for money transferring. However, this does not limit the future developers that will set hands on the code to add messaging or opening new account features, even their own security measures between user and application. This flexibility of knowing what's inside and the option to add, remove or customize features to one's preferences, is what makes the API we present a huge innovation among banking interfaces.

Chapter 10

Conclusion

In this work we have discussed about the failures in banking systems and methods to improve them. Further we took the discussion to mobile devices and show that even a non-rooted mobile device is in danger by numerous attacks. Moving ahead from the banking failures and mobile device threads, we presented the methods employed by banks to protect the mobile devices from fraud utilizing various authentication methods. Then, we presented five case studies of software token authentication techniques, two open-source and three through reverse engineering work.

Moving ahead to our vulnerability assessment work, we manage to get the insight of user-server communication for online and mobile platforms of HSBC. At that point we show the vulnerabilities found on the bank's webpage and the lack of usability for mobile applications, accompanied with bad reviews from Google store. Besides inspecting the online and mobile application, we took a deeper dive and understood how OTP work for HSBC log-on and transaction options which we present and discuss in section 7.

Finally, we introduce our own open API that is able to log-on, view transaction, history and make transactions, without involving any HSBC applications since OTP is also a feature of our application. On evaluation we show that our application, outperforms HSBC in delays and data usage. Furthermore, our open API does not send data to third parties but only what is necessary to perform the tasks that the user needs and since the code is available to read anybody is able to understand what happens behind the interface and add/modify features as he likes.

This work could be an example of a minimum requirement open API after PSD2 law is applied. In addition, by identifying some vulnerabilities of the actual bank's application proves that R. Anderson's approach for improving banking systems could really apply on today's society. Having an open API will provide valuable feedback and improvement from the community that will contribute to a safer banking experience for customers.

Bibliography

[1] *Cyprus banking system collapse*
Online: `http://www.businessinsider.com/r-amid-fears-of-greek-controls-cyprus-shows-restrictions-are-bearable-2015-6`

[2] J. Pouwelse and C. Kyprianou, *Vulnerability Analysis of Banking Applications on Android Smartphones*, 2017

[3] *The Stagefright exploit: What you need to know:*
Online: `http://www.androidcentral.com/stagefright`

[4] *TROJAN-BANKER.ANDROIDOS.SVPENG:*
Online:
`https://threats.kaspersky.com/en/threat/Trojan-Banker.AndroidOS.Svpeng/`

[5] C. Castillo, *Phishing Attack Replaces Android Banking Apps With Malware:*
Online: `https://securingtomorrow.mcafee.com/mcafee-labs/phishing-attack-replaces-android-banking-apps-with-malware/`

[6] *The Biggest Splash at BlackHat and DEFCON 2015:*
Online: `http://blog.zimperium.com/the-biggest-splash-at-blackhat-and-defcon-2015/`

[7] J. Pouwelse, L. Versluis and M. Vos, *Internet-of-Money: sub-second money routing using existing bank accounts*, 2016

[8] J. Awsome and J. Pouwelse, *A vulnerability analysis on mobile banking applications*, Unpublished research, 2015

[9] J. Doe and J. Pouwelse, *A vulnerability analysis of smartphone banking applications*, Unpublished research, 2015

[10] *Rogue APs: All you need to know about them:* Online:
`http://www.rogueap.com`

[11] *Man-in-the-middle attack:*
Online: `https://www.owasp.org/index.php/Man-in-the-middle_attack`

[12] R. Anderson, *Why Cryptosystems Fail*, 1st Conf.- Computer and Comm. Security '93, Nov. 1993

[13] S21sec Labs, *ALICE: SIMPLICITY FOR ATM JACKPOTTING*, Jan. 2017
Online:
`https://www.s21sec.com/en/blog/2017/01/alice-simplicity-for-atm-jackpotting/`

[14] M. Brignall, *So you think you're safe doing internet banking?*, the guardian, Nov 2015
Online: `https://www.theguardian.com/money/2015/nov/21/safe-internet-banking-cyber-security-online`

[15] M. Brignall, *Why are Britain's banks blaming customers for online banking fraud?*, the guardian, Aug. 2017
Online:
`https://www.theguardian.com/commentisfree/2017/aug/08/banks-online-banking-fraud-ross-mcewan-rbs`

[16] Payment services (PSD 2) - Directive (EU) 2015/2366:
Online: `https://ec.europa.eu/info/law/payment-services-psd-2-directive-eu-2015-2366_en`

[17] *HSBC website:*
Online: `www.hsbc.co.uk`

[18] *Entrust IdentityGuard One-Time Passcodes (OTP)*
Online: `https://www.entrust.com/products/mobile-otp/`

[19] *VASCO DIGIPASS:*
Online: `https://www.vasco.com`

[20] Michael Bentley, *Lookout discovers new trojanized adware; 20K popular apps caught in the crossfire*,
Online: `https://blog.lookout.com/trojanized-adware`

[21] *Android Debug Bridge*
Online:
`https://developer.android.com/studio/command-line/adb.html`

[22] *Androguard documentation:*
Online: `http://doc.androguard.re/html/index.html`

[23] *Androguard Reverse engineering:*
Online: `http://www.technotalkative.com/part-1-reverse-engineering-using-androguard/`

[24] *A.R.E-Android Reverse Engineering:*
Online:
`https://redmine.honeynet.org/projects/are/wiki`

[25] *Santoku:*
Online: `https://santoku-linux.com/download`

[26] *Smali wiki:*
Online: `https://github.com/JesusFreke/smali/wiki`

[27] *Beginer's guide to smali:*
Online: `https://forum.xda-developers.com/showthread.php?t=2193735`

[28] *Smali example:*
Online: `http://androidcracking.blogspot.nl/2010/09/examplesmali.html`

[29] *Simplify:*
Online: `https://github.com/CalebFenton/simplify`

[30] *JMD*
Online: `https://github.com/Contra/JMD`

[31] *Apktool documentation:*
Online: `https://ibotpeaches.github.io/Apktool`

[32] *Signing android app:*
Online:
`https://developer.android.com/studio/publish/app-signing.html`

[33] *Dex2jad:*
Online: `https://github.com/pxb1988/dex2jar`

[34] *JD-GUI:*
Online: `http://jd.benow.ca/`

[35] *Android A to Z: What is Dalvik:*
Online: `http://www.androidcentral.com/android-z-what-dalvik`

[36] *JDB-java debugger:*
Online: `http://docs.oracle.com/javase/7/docs/technotes/tools/windows/jdb.html`

[37] *IDA Pro:*
Online: `https://www.hex-rays.com/products/ida/ida-executive.pdf`

[38] *Debugging Dalvik programs with IDA:*
Online: `https://www.hex-rays.com/products/ida/support/tutorials/debugging_dalvik.pdf`

[39] *Drozer:*
Online: `Online:https://labs.mwrinfosecurity.com/tools/drozer/`

[40] *Drozer data leakage attack:*
Online: `https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-drozer-user-guide-2015-03-23.pdf`

[41] *Infosec: Side Channel Data Leakage:*
Online:
`http://resources.infosecinstitute.com/android-hacking-security-part-4-exploiting-unintended-data-leakage-side-channel-data-leakage/#article`

[42] *Infosec resources:*
Online: `http://resources.infosecinstitute.com`

[43] *Projects/OWASP Mobile Security Project - Top Ten Mobile Risks 2016:*
Online: `https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10`

[44] *Android: Saving Key-Value Sets*
Online: `https://developer.android.com/training/basics/data-storage/shared-preferences.html`

[45] *Hacking an Android Banking Application*, Juliano Rizzo, 2012
Online: `http://blog.ioactive.com/2012/11/hacking-android-banking-application.html`

[46] *Secure Preferences:*
Online:
`https://github.com/scottyab/secure-preferences`

[47] *Android: Saving Data in SQL Databases*
Online: `https://developer.android.com/training/basics/data-storage/databases.html`

[48] *Zetetic - SLCipher:*
Online: `https://www.zetetic.net/sqlcipher/`

[49] *Infosec: Insecure local Storage* Online: `http://resources.infosecinstitute.com/android-hacking-security-part-10-insecure-local-storage/`

[50] *Why NoSQL trumps relational databases for mobile applications*, Online: `https://www.techopedia.com/2/29256/development/mobile-development/why-nosql-trumps-relational-databases-for-mobile-applications`

[51] *What is android ADB backup?* Online: `https://forum.xda-developers.com/showthread.php?t=2011811`

[52] *infosec: Broken Cryptography* Online: `http://resources.infosecinstitute.com/android-hacking-security-part-16-broken-cryptography/`

[53] *What can be done to promote trust in electronic banking systems?* Online: `http://www.techradar.com/news/world-of-tech/what-can-be-done-to-promote-trust-in-electronic-banking-systems-1280633/2`

[54] SecurEnvoy, *What is 2FA?*, Online: `https://www.securenvoy.com/two-factor-authentication/what-is-2fa.shtm`

[55] *Online banking and safety: TAN Methods* Online: `http://mkenyaujerumani.de/2014/02/07/online-banking-and-safety-tan-methods/`

[56] RedTeam Pentesting GmbH, *Man-in-the-Middle Attacks against the chipTAN comfort Online Banking System*, Online: `https://www.redteam-pentesting.de/publications/2009-11-23-MitM-chipTAN-comfort_RedTeam-Pentesting_EN.pdf,2009`

[57] *List of websites and whether or not they support 2FA*, Online: `https://twofactorauth.org/`

[58] A. Wiesmaier, M. Fischer, M. Lippert, J. Buchmann, *Outflanking and Securely Using the PIN/TAN–System*, Proceedings of the 2005 International Conference on Security and Management (SAM'05), June 2005

[59] C. Serrato, *Soft tokens, hard security: the best way to beat online banking threats*,

Online: `http://www.techradar.com/news/world-of-tech/soft-tokens-hard-security-the-best-way-to-beat-online-banking-threats-1284689/2` , 2015

[60] S. Musil, *Zeus botnet steals $47M from European bank customers*, Online: `https://www.cnet.com/news/zeus-botnet-steals-47m-from-european-bank-customers`, 2015

[61] B. Mueller, *Hacking Soft Tokens Advanced Reverse Engineering on Android*, 2016

[62] *OATH website:* Online: `https://openauthentication.org`

[63] R. Moulds, *What is PKI and why is it important?* Online: `https://www.thales-esecurity.com/blogs/2013/march/what-is-a-pki`, 2013

[64] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley and W. Polk, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, Online: `https://www.ietf.org/rfc/rfc5280.txt,2008`

[65] *ETSI - SIM:* Online: `http://www.etsi.org/technologies-clusters/technologies/smart-cards/sim`

[66] D. M'Raihi, S. Machani, M. Pei and J. Rydell, *TOTP: Time-Based One-Time Password Algorithm*, Online: `https://tools.ietf.org/html/rfc6238`, 2011

[67] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache and O. Ranen, *HOTP: An HMAC-Based One-Time Password Algorithm*, Online: `https://www.ietf.org/rfc/rfc4226.txt` , 2005

[68] T. Valverde, *Reverse engineering mt bank's security token*, Online: `http://blog.valverde.me/2014/01/03/reverse-engineering-my-bank's-security-token/`

[69] *RSA SecureID:* Online: `https://www.rsa.com/en-us/products/rsa-securid-suite`

[70] *ProGuard:* Online: `https://www.guardsquare.com/en/proguard`

[71] *Stoken - Software Token for Linux/UNIX:* Online: `https://github.com/cernekee/stoken`

[72] *Frida:*
Online: https://www.frida.re/docs/android/

[73] *Stellaris LaunchPad:*
Online: http://www.ti.com/stellaris-launchpad

[74] *Cryptosuite:*
Online: https://github.com/Cathedrow/Cryptosuite

[75] *RTClib*
Online: https://github.com/adafruit/RTClib

[76] *Paym:*
Online: https://www.paym.co.uk

[77] K. Hall, Day 2: Millions of HSBC customers still locked out of online banking, Jan 2016
Online: https:
//www.theregister.co.uk/2016/01/05/millions_of_
hsbc_customers_still_locked_out_of_online_banking/

[78] *Fiddler:*
Online: http://www.telerik.com/fiddler

[79] *Requests: HTTP for Humans*
Online: http://docs.python-requests.org/en/master/

[80] *Mozila trust store:*
Online:
https://hg.mozilla.org/mozilla-central/raw-file/
tip/security/nss/lib/ckfw/builtins/certdata.txt

[81] *An introduction to JEB anrdoid debuggers:*
Online: https://www.pnfsoftware.com/blog/jeb-
android-debuggers

[82] C. Collberg, C. Thomborson, D. Low, *A taxonomy of obfuscating transformations*, Technical Report 148, Dept. of Computer Science, Univ. of Auckland, 1997.

[83] L. Hardesty, MIT news, Nov 2015,
Online: http://news.mit.edu/2015/data-transferred-
android-apps-hiding-1119

[84] *Liveperson url:* https://www.liveperson.com/

[85] *APPDYNAMICS product overview:* , Online:
https://www.appdynamics.com/product/