Let's Talk Money: Evaluating the Security Challenges of Mobile Money in the Developing World

Sam Castle
University of Washington
stcastle@cs.washington.edu

Fahad Pervaiz
University of Washington
fahadp@cs.washington.edu

Galen Weld
Cornell University
gcw33@cornell.edu

Franziska Roesner
University of Washington
franzi@cs.washington.edu

Richard Anderson
University of Washington
anderson@cs.washington.edu

ABSTRACT

Digital money drives modern economies, and the global adoption of mobile phones has enabled a wide range of digital financial services in the developing world. Where there is money, there must be security, yet prior work on mobile money has identified discouraging vulnerabilities in the current ecosystem. We begin by arguing that the situation is not as dire as it may seem—many reported issues can be resolved by security best practices and updated mobile software. To support this argument, we diagnose the problems from two directions: (1) a large-scale analysis of existing financial service products and (2) a series of interviews with 7 developers and designers in Africa and South America. We frame this assessment within a novel, systematic threat model. In our large-scale analysis, we evaluate 197 Android apps and take a deeper look at 71 products to assess specific organizational practices. We conclude that although attack vectors are present in many apps, service providers are generally making intentional, security-conscious decisions. The developer interviews support these findings, as most participants demonstrated technical competency and experience, and all worked within established organizations with regimented code review processes and dedicated security teams.

Categories and Subject Descriptors

K.4.4 [**Computers and Society**]: Electronic Commerce—*security*

Keywords

ICTD; mobile money; fraud; Android; finance; mBanking; human factors

1. INTRODUCTION

Digital financial services (DFS) constitute a rapidly growing industry that provides access to formal financial instruments through mobile technology. These services operate on existing mobile networks and are managed by telcos, banks, and third-party software companies. One critical aspect is that these services allow individuals with no formal financial history to establish an account, often without needing to travel to a physical bank location.

The basic services offered by DFS applications include monetary deposits, withdrawals, and person-to-person transactions. Many include additional value-added services, such as government-to-person payments, loans, and payments for goods and utilities. Deposits and withdrawals are almost always handled by a network of agents, typically shop owners or community members, who are employed by the service operator. They accept cash deposits in exchange for digital currency transferred to the user's account, and they keep stores of cash for users to withdraw their digital balances.

In this work, we focus on security vulnerabilities in current DFS deployments. We begin by scoping our analysis within a threat model, and then we analyze a set of existing products. To address the potential vulnerabilities uncovered in our research and in prior work, we examined the current development process by interviewing app developers. Specifically, our contributions are threefold:

- We systematically define a threat model to outline potential attacks on DFS applications, and we use this model to inform future design decisions and as an evaluative framework for security analysis.
- We conduct an in-depth security analysis on 397 current DFS deployments, including 197 Android apps, and we consider a large number of factors, which have not been studied in detail previously for DFS products.
- We interview 7 developers and project managers. These interviews offer convincing insights into current practices and noteworthy issues for further research.

2. MOTIVATION

Strong security and privacy measures are critical to expanding DFS products to the world's poor and unbanked. For a person who has been living their entire life with structured financial institutions, falling victim to a security failure may permanently divert them from that particular bank or service. For a person with no former structured financial experience, running into a security failure may deter them from formal financial systems as a whole. For these reasons, we consider security and privacy issues to be especially paramount in the context of the developing world.

A recent analysis on mobile money by Reaves et al. [28] showed that "the majority of these apps fail to provide the protections needed by financial services. . . threatening to erode

ACM DEV '16 November 17-22, 2016, Nairobi, Kenya

© 2016 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4649-8/16/11.

DOI: http://dx.doi.org/10.1145/3001913.3001919

trust in branchless banking and hinder efforts for global financial inclusion." We consider these claims below.

Response to Prior Work. In their review of vulnerabilities in existing Android apps, Reaves et al. [28] consider four broad categories: SSL/TLS & certificate verification, non-standard cryptography, access control, and information leakage. We consider each of these in turn.

SSL/TLS certificate verification along with non-standard cryptography can be remedied by correctly configuring SSL/TLS on the server and using standard encryption algorithms. We dig deeper into these possibilities in Section 5, where we present several case studies in which developers are at the mercy of business partners who manage the server setup or encryption methods. The main takeaway here is that from a programmer's perspective, viable solutions to these issues exist, and in our experience, developers are aware of best practices, but the primary bottleneck may be institutional factors, such as differing priorities among partner organizations.

The next point, access control, revolves around user authentication protocols and communicating user credentials. The mentioned vulnerabilities are mainly the result of improper encryption or of communicating sensitive information over an insecure channel. Weak password requirements, on the other hand, are a concern, but our app analysis found that many apps are defending against related attacks by blocking an account after 3 to 5 failed password attempts. The failed attempt restriction guards against brute-force attacks on all possible passwords. In Section 4.5, we provide a more detailed analysis of the existing app ecosystem.

Information leakage concerns private user data, which can be compromised either during client-server communication or via naive storage on the client device. Reaves et al. explain that apps write sensitive data in private logs, mostly for debugging purposes, but these logs can be accessed by any other app with the `READ_LOG` permission in Android versions older than Version 4.1.

Although this is a serious potential threat for data leakage, it is arguably not realistic in practice. Leakage from private logs can only happen in limited cases because only 4% of Android devices currently run these older, vulnerable versions [1]. The majority of apps we studied, however, still allow older versions and thus fail to completely shut off this vulnerability (Figure 1). As we discuss in more detail in Section 4.2, most apps are not at risk to this issue—our app analysis identified only two apps that write data to a file.

Android. In this work, we focus on mobile applications built for Android. A majority of prior work in security for mobile devices has focused on the Android platform, and Android seems to dominate the market share in the regions where our work is focused—primarily Africa, South America, and Southern Asia—as evidenced by relatively low usage of the iOS default browser, Safari, on mobile devices [3]. Additional research is needed to generalize claims to other mobile platforms, though studies have shown that other systems encounter similar problems to those in Android [11,16].

3. THREAT MODEL

Before assessing security vulnerabilities, it is essential to understand the range of possible attacks and potential adversaries. This process, known as threat modeling, is common within the computer security community. Having an understanding of potential security threats helps developers choose which security features to implement and allows researchers and quality assurance teams to ground their security analyses in reality. In this section, we enumerate a wide range of attacks that could compromise the security of mobile banking applications. This threat model is focused on DFS in a developing world context and is based on information from academic research [9,10,13,28,29], industry reports [4,19,21], real-world exploits [5–7], and our own interviews with developers.

We consider this threat model as a contribution in itself because it incorporates a diverse array of information into a systematic, novel documentation of possible attacks specific to the context of DFS within the developing world. We expand upon previous work, with this threat model process being most similar to work from Cobb et al. [13], who examined data collection processes in the developing world. This model is a theoretical collection of *possible* attacks and adversaries, and thus it is a *superset* of the actual attacks that occur in deployed applications. When considering each risk, computer security is often about managing tradeoffs between total protection and other factors, including usability and resources. This threat model identifies the possible risks, and our later sections on app analysis and developer interviews will inform understanding of both the likelihood and consequence of those risks.

Security Goals. In this work, we consider both attacks on users and attacks on organizations. User security has been the primary focus of previous work [28], but in our interviews with developers, we found that it is necessary to consider the importance of protecting company assets.

Computer security goals are often modeled around the "CIA" triad: confidentiality, integrity, and availability. Confidentiality is akin to privacy, meaning the protection of any sensitive or identifying information about a customer, such as biometric data, account balance, and passwords. Integrity refers to the accuracy and trustworthiness of data—if a user initiates a transaction to pay a merchant, it is important that the correct amount of money is sent to the correct recipient. Availability is the need for access to services within a reasonable time frame; for example, a customer's money should be available for withdrawal when they need it.

Potential Adversaries. We identify actors who may be motivated to compromise any of the above security goals.

- *Customers* or other company outsiders may attempt to steal money or information from unsuspecting users, or they may steal money and services from the organization itself. For attacks on users, *friends and family members* pose unique security challenges because they may have access to personal information, such as the user's password, device, or account recovery details.
- *Agents* act as intermediaries between users and their accounts, and agents can abuse this position in ways such as charging extra fees or secretly depositing money into their own accounts.
- *Organization employees,* beyond agents, may have access to confidential information, such as user account credentials or lingering security vulnerabilities. A rogue employee may use this knowledge to steal money or sell sensitive information to other adversaries.

Potential Attacks. Table 1 summarizes concrete possible attacks by which potential adversaries may try to compromise the above security goals.

	Attack Name	*Description*
Confidentiality	**External Apps**	In early versions of Android, apps can read private data stored by other apps on the same device.
	External Libraries	Developers often include 3rd-party libraries in their applications for social media, advertisements, cryptography, etc. Such libraries can introduce unintentional vulnerabilities or actively-malicious code, and researchers have found that legitimate libraries may be duplicated and repackaged with malware [11]. Advertising and analytics libraries are also known to track user data [14].
	SMS Intercept	When apps communicate sensitive data via SMS, adversaries can intercept the SMS to learn private information about an individual or to take control of a user's account activity. SMS communications have known vulnerabilities [23], and Android has known issues in communications between apps [12].
Integrity	**Server Attack**	An adversary gains unauthorized access to the service's server. This includes full root exploits as well as gaining access to partial server logs, database information, or proprietary source code.
	Man-in-the-Middle (MITM)	An adversary is able to intercept network traffic between the client and server. This allows the adversary to observe any transmitted information and to also send fake data to either party.
	Authentication Attack	There are many ways in which an adversary can gain unauthorized access to a user's account. These attacks are facilitated by services with unlimited login attempts, weak password reset procedures, and accounts where the user ID is the phone number, which is often considered public information.
	SMS Spoof	This fraud occurs when a service uses SMS to communicate with users. When used for receipts, one user can send an SMS to another user to "confirm" a transaction, when in fact no money has been transferred. Fraudsters may also be able to pose as the organization to engineer phishing attacks.
	Agent-driven Fraud	Many people, due to illiteracy or general fear of making a mistake, trust agents to process transactions on their behalf [21]. This enables a variety of attacks from agents targeting customers, such as stealing money intended for deposits (SMS spoofing) or charging additional, unlawful fees. Customers can also defraud agents with counterfeit currency or physical force.
	Fake Accounts	If new accounts are easy to obtain, fraudsters will have more opportunities to create disposable accounts for scams. Attacks which rely on fake accounts can be mitigated by strict ID requirements for account setup and a system for reporting and disabling fraudulent accounts.
Availability	**Data Loss**	Rather than gaining access to sensitive information, the adversary destroys or corrupts business data. This may range from a complete database wipe to erasing the data of a single user.
	Denial-of-Service (DoS)	Targeting a service's connection to the server with useless traffic can block actual traffic from reaching the server. This is an attack on both the service provider and the customer—the organization loses revenue and reputation, and the customer cannot access their account.
	Theft of Services	Customers can target organizations by gaining free use of services that would normally require payment. For example, apps may include zero-rated URLs, which can be extracted by tech-savvy individuals to bypass telco data walls and browse on the web without paying for a service bundle.
	Device Theft	If an adversary is able to steal a user's physical device, then the thief may be able to gain access to funds or private information. Some services bind accounts to a user's device in order to bypass password login procedures, which would allow adversaries to easily compromise accounts on lost or stolen devices.

Table 1: Hypothetical attacks on mobile money apps.

4. APP ANALYSIS

In this section, we characterize security-related features of current apps in the market in terms of their potential for information leakage, their requested permissions, and their inclusion of 3rd-party URLs. Then, we conduct a manual analysis of websites for 71 different services.

4.1 Methodology

To build a comprehensive list of existing digital financial services, we started with the GSMA mobile money deployment tracker [18]. This database includes "mobile-enabled products and services in the developing world," and it is based on data provided by mobile network operators directly to the GSMA. As of March 2016, the database contained 284 deployments, though some are similar services offered by the same parent company but repackaged for different countries. We consider these services as unique because during our interviews with developers, we found that regulations in different countries require unique considerations. We used this database as a starting point and also searched the web and the Google Play Store for additional deployments.

We amassed a database of 397 services. Many of these services operate only through USSD, but we found 210 that have associated Android apps. Of these apps, 206 had available metadata on the Google Play Store, and Figure 2 shows these 206 apps sorted by their total number of downloads.

Due to country-specific device compatibility, we were unable to download 13 of the 210 Android apps, so our large-scale analysis considers only these 197 apps. After downloading the apps, we decompiled the apks using Apktool [2], a tool for reverse engineering 3rd-party, closed, binary apps.

4.2 Information Leakage

In Section 2, we discussed a vulnerability identified by Reaves et al. [28] in which sensitive data, logged by the app for debugging purposes, is printed to Android's log and can be accessed by any other app on the device. This problem was directly addressed by Android system updates, and in Android Version 4.1 or higher, an app's debug log files can only be accessed by the app itself. At the time of this writing, 96% of active Android devices, defined by those which accessed the Google Play Store in the previous week, are operating with Android Version 4.1 or higher [1]. In the context of emerging markets, this statistic is unknown.

Android apps in the Google Play marketplace must specify the lowest version of Android on which the app can run.

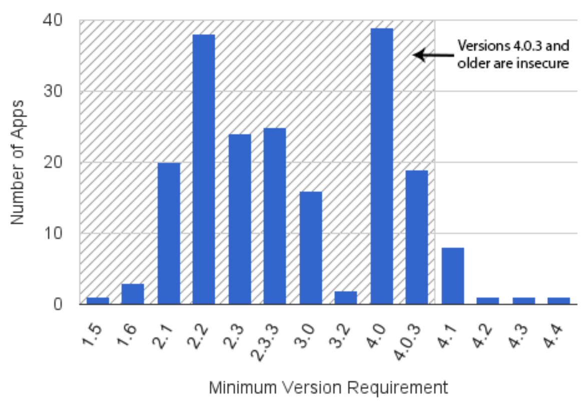

Figure 1: The minimum Android OS requirements for the 206 apps. 8 apps specify requirements that vary by device and are not depicted in this graph.

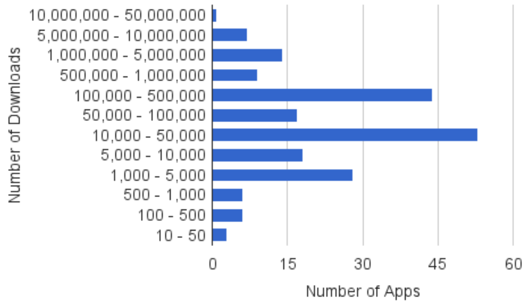

Figure 2: The number of downloads by app for the 206 apps with available data.

To really close the door on this vulnerability, apps should elect to support only Version 4.1 or higher; however, only 11 (5%) of the 206 Android apps we studied enforce this version requirement. A breakdown of the apps we studied and the minimum Android OS version they support is shown in Figure 1. Perhaps a greater prevalence of older phones in emerging markets encourages service providers to offer compatibility, but this practice leaves more room for mistakes.

An alternate way that apps may leave data on the device is by explicitly writing data files to storage. These files are permanent, unlike logs that are ultimately overwritten as new data is logged. Insecure apps may write these data files using the Android shared preference `MODE_WORLD_WRITEABLE` that writes data to the SD Card and makes it accessible by all apps on the device. All apps should use `MODE_PRIVATE` for storing private data. Conversely, this issue is moot if an app simply does not log or write any data. During our interviews, many developers claimed that their apps do not store any data on the device. Of the 197 apps we surveyed, 1 app uses `MODE_WORLD_WRITEABLE`, and 1 app uses `MODE_PRIVATE`.

4.3 Permissions

By decompiling `AndroidManifest.xml` files, we cataloged the permissions requested by each of the 197 apps. As shown in Figure 3, many of the permissions are to be expected, such as Internet and SMS access. SMS responses are commonly used for balance inquiries or receipts, though the relatively coarse granularity of the Android permission system gives these apps unfettered access to all SMS messages on the phone. Additionally, SMS communications can be intercepted over the network [23], from other apps on the device [12], or by intrusive people with access to the device.

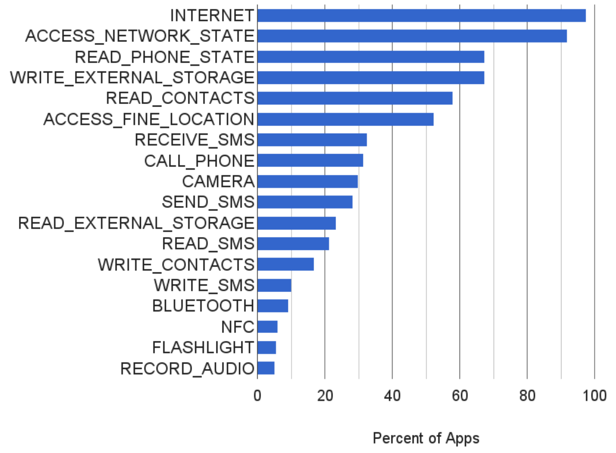

Figure 3: A subset of the permissions requested by the 197 Android apps we were able to download.

65% of the apps request permission to write data to external storage; this provides an upper bound on the potential for information leakage described in the previous section.

Other permissions are more questionable—it is unclear why a banking app would require access to a device's flashlight or microphone. The prevalence of over-privilege in Android apps has been previously documented [17], but we did not investigate how the apps use these permissions, so it is unknown whether each app's permissions are used for legitimate functionality, an unusual practice, or not used at all. One possibility, based on prior research [27], is that external advertising libraries are a cause of excessive permissions.

4.4 URLs and 3rd-Party Libraries

Apps communicate to servers for a variety of reasons. For one, financial apps must send data to their own servers, yet they also may include links to company pages, social media pages, ad servers, analytics trackers, and more.

Figure 4 shows the use of HTTP and HTTPS URLs within the 197 Android apps. Half of all apps included more than 20 HTTPS URLs and more than 25 HTTP URLs. Our analysis statically searched for URL strings in the decompiled code, and we did not distinguish between URLs a user would need to click on versus URLs that the app is definitely contacting.

All apps studied included HTTP URLs within the apps. This opens the door to possible man-in-the-middle (MITM) attacks by attackers in control of the network (e.g., a compromised router). If a user clicks a link that opens a page over HTTP, a MITM can substitute any other HTML page, such as a fake login page to request the user's PIN or password. This style of phishing attack is especially problematic on mobile devices because previous research has shown shortcomings with Android security indicators [15].

About 10% of the apps did not include any HTTPS URLs. Assuming these apps are transmitting sensitive information, this indicates that these apps are sending this information over HTTP connections and are possibly using custom encryption algorithms. Custom encryption regularly leads to mistakes [28], and in these cases, a passive MITM could gain control over all communications. Additionally, common au-

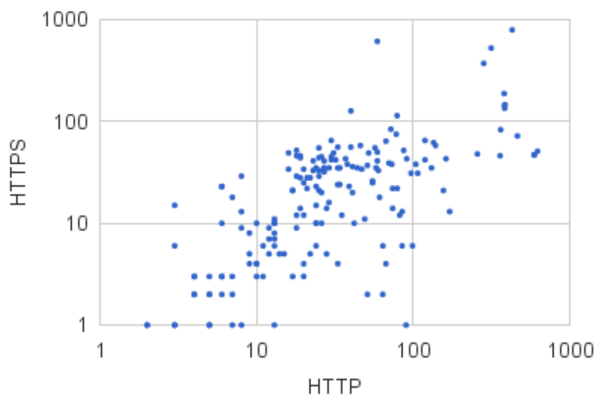

Figure 4: Distribution of HTTPS versus HTTP URLs in each of the 197 apps studied. Each point represents a single app.

tomated tools, such as MalloDroid [15], for detecting vulnerabilities in SSL/TLS connections would not flag the app because the apps are not misusing SSL—they are just not using it, an issue previously identified by Reaves et al. [28].

A large number of apps include links to common social media sites, and the most common social media URLs across apps are shown in Figure 5. This presents a potential hidden privacy violation because these platforms may track user information, including location, unique device ID, and other identifying details [14]. Users may be unaware of tracking, and user perception of privacy concerns over tracking are not well understood in the developing world.

Other tracking and advertising libraries, shown in Figure 6, were also fairly common. All of the sites in Figure 6 are libraries which have been shown to require additional Android app permissions and track sensitive user data [27]. These libraries also present a security concern because they include 3rd-party code not written by the original app developers. A malicious or buggy library can introduce new data leaks and security vulnerabilities into any app. In prior research, Chen et al. [11] characterized the prevalence of these attacks, and in particular, they exposed harmful variations of the `media.admob.com` and `phonegap.com` libraries.

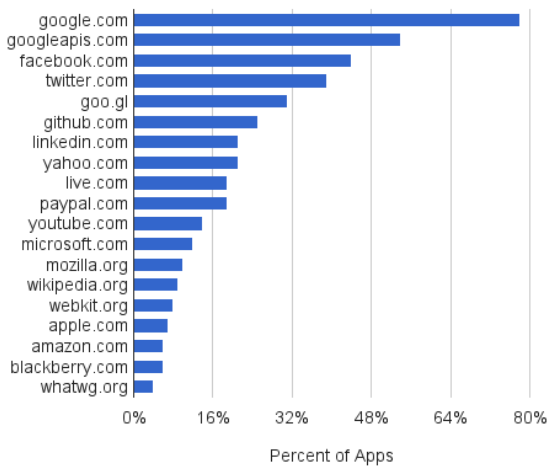

Figure 5: Most common social media URLs in apps.

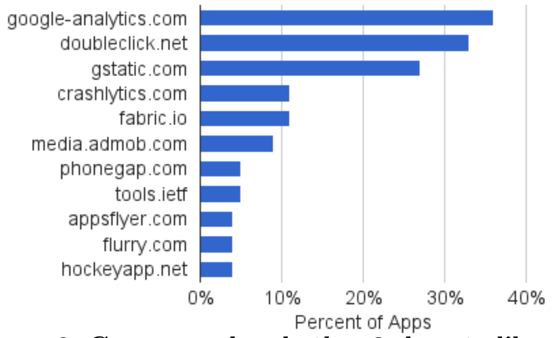

Figure 6: Common ad and other 3rd-party libraries.

4.5 In-Depth App Analysis

In this section we describe an in-depth manual analysis of 71 digital financial service deployments, including the 29 Android apps with over 500,000 downloads and a random subset of services with Android or USSD-only deployments. We collected the information about these services by studying their websites and other public, promotional material. Many of these services provide only rudimentary information on their websites, and several had inaccessible or untranslatable websites, yet we were still able to compile meaningful aggregate information on important aspects of their security practices. A summary of results is shown in Figure 7, and details are given below. Information for all 71 services was not always available, as specified in the descriptions below.

Organization Type. 11 were produced by banks or other financial institutions, 43 were produced or operated by cellular carriers, and 17 were produced by 3rd-party software companies that are neither affiliated with a bank nor a telco.

SMS Responses. Many USSD-based services provide SMS messages to their users in response to USSD or app-initiated queries for current balance, transaction history, or other similar financial information. As SMS has known vulnerabilities [23], these SMS responses are a potential source of user's financial information for malicious parties. Of the 56 apps that listed relevant information on their website, 36 of them returned account information, typically balance and recent transactions, via SMS. One app even allows balance queries to be initiated via SMS without any USSD operation. Not only are SMS messages readable by any person with access to the phone, they can also potentially be intercepted over the network [23] or accessed by other apps on the device [12].

Password Reset Information. Of the services analyzed, there was a wide range of password reset processes and a diversity of information required to access an account with a forgotten password or PIN. 34 services provided information on password reset procedures in their documentation; of these, 14 required users to call customer service to verify their account information and identity. 4 required users to visit an agent in person in order to recover their account. The rest either provided in-app or in-browser password reset options, which typically required the user to answer security questions or provide personal information listed at the time of account creation. Adversaries may have access to some of this information, such as the user's address or phone number. 4 apps reset the user's password to the default of '1234' and then request that they change it—a clear opening for a malicious individual to gain control of an account.

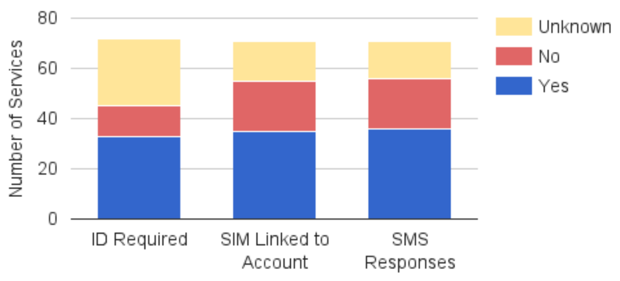

Figure 7: A summary of information for the 71 services analyzed in depth.

	Location	Org.	Org. size	App downloads	Dev?
P1	Nigeria	telco	> 500	50k–100k	yes
P2	Nigeria	bank	> 500	100k–500k	yes
P3	Kenya	bank, software	50–100	5k–10k	yes
P4	Uganda	software	100–500	varies w/ partner	yes
P5	Zimbabwe	telco	> 500	50k–100k	yes
P6	Colombia	bank	> 500	50k–100k	no
P7	Colombia	bank	> 500	50k–100k	yes

Table 2: Summary of Interview Participants.

SIM Coupled to Account. Of the 67 mobile money services with available information, 35 only allowed the user to access their account via a device with the user's SIM card. This forces malicious actors to have physical access to their target's device in order to access the account, yet it also limits flexibility for legitimate account holders.

ID Requirement. 32 of the services analyzed required the user to present a government-issued ID to create a mobile money account. One service required that the user's fingerprints be taken in order to open an account, and the remainder did not have any such ID requirement, although many of these require a cellular account, and it is possible that cellular carriers require ID at the time of account creation.

Account-less Money Transfers. At least 9 of the services analyzed permit users to transfer money to individuals without an account on any mobile banking platform. The sender generates a single-use voucher code that can be sent to the recipient via SMS and then exchanged with an agent for cash. While convenient, these account-less transfers expose an additional vulnerability—if an adversary accesses a voucher code, then they will easily be able to steal the funds, as the identity of the recipient is not verified by the agent.

5. DEVELOPER INTERVIEWS

Our analysis in Section 4 helps characterize the current state of affairs in DFS application security, but such an analysis is limited in its ability to identify the root causes of vulnerabilities. Based on the potential vulnerabilities identified in this research and prior work, one hypothesis is that apps are created by unskilled developers through informal, remiss processes. To improve security practices, researchers have suggested better training for developers or better external libraries which enable easier implementation of secure algorithms [16]. Such solutions offer promising tools, but without understanding the development model, these suggestions may not be able to address the true limitations.

With this goal in mind, we gained an impression of the existing ecosystem by interviewing 7 current employees of DFS providers. Of these, 6 interviewees work directly on writing product code, and the additional interviewee writes design specifications. The interviewees represent a diverse set of countries, located primarily within Africa, and work for several different types of organizations. A summary of information about the participants is listed in Table 2.

These interviews are not intended to draw comparisons to a presumed "best" or "Western" approach to application development, as we have no baseline for comparison to the developed world, and the specific demands and constraints are different in emerging markets. Rather, these interviews are intended to provide glimpses into the current app development ecosystem and to promote topics for further consideration. Our findings from the interviews serve as evidence that the developers are operating within a complex ecosystem, and many issues are the result of a combination of unique regional factors, such as government regulations, bank specifications, or budget limitations.

5.1 Recruitment

To recruit participants, we used our catalog of active deployments, described in Section 4.1, to find 249 unique email addresses related to mobile money products. We sent an email to each of these addresses to briefly introduce our research and to invite recipients to complete a short survey.

The primary purpose of the initial survey was to collect basic demographic information and to request a follow-up interview. Beyond contributing to our research goals, there were no incentives for respondents to either complete the survey or participate in an interview. From the email survey, we recruited 5 participants. The remaining two interviewees (P2 and P7) were recruited through contacts of P1 and P6, respectively. We received approval for this entire process through our institution's human subjects review board.

5.1.1 Limitations

Given the limited sample size of interviewees, our research outcomes are primarily to introduce new considerations and points of interest. We attempted to reduce bias in recruitment by not mentioning security explicitly before the interview. It is worth noting that all participants work for fairly large organizations. Similarly, we recruited participants via email, which biases our results toward organizations whose apps had actively-monitored email addresses. Accordingly, our interview results reflect a certain class of developers.

5.2 Interview Process

The interviews, conducted remotely, lasted about 45 minutes and began by discussing general information about the participant's current organization, job responsibilities, and work environment. Topics then included the participant's education and experience, training offered by the organization, and the code development and review processes. The majority of the interview was spent discussing security, including the organization's policies, the participant's technical expertise, and the participant's mental threat model.

The two primary authors were present for every interview. One researcher acted as the main interviewer while the other recorded notes. Each of the authors reviewed all notes and met to discuss themes and lessons that arose. These themes are discussed in the following section.

5.3 Interview Results

Every developer we interviewed exhibited basic technical competence, held a degree in a relevant field, and worked within an established corporation with standard code review processes and dedicated security teams. We begin this section by describing these positive findings. Then, we identify several possible reasons why vulnerabilities still persist within the app ecosystem, and we propose several concrete changes to improve security practices.

Table 3 shows a high-level overview of our findings from the interviews. The results from participants P6 and P7 have been combined because P6 and P7 are both veteran employees in the same organization. All overlapping information was consistent between the two interviewees.

5.3.1 Overall Positive Impressions

As shown in Table 3, every developer we interviewed was working within a corporate structure with established code review processes, and all but one developer had extensive experience and received specific technical training from their organization. The developers also had strong education backgrounds, as every developer held a degree in a relevant field.

Five developers indicated that their organizations have dedicated security experts within separate quality assurance teams. Similarly, many organizations had strict vetting processes as a security measure for external libraries. These findings demonstrate that security is a priority, and all participants, except for P7, stressed the role of security in their work. P5 reflected, "security is really important. We are very strict about it. We want to make sure developers make all of our applications secure."

Similarly, P6, who is not a developer but a product designer, described security as "the responsibility of the entire team." P6 added that the organization makes a concerted effort to train its employees by offering courses "whenever there are new ideas in the market." P1 emphasized the importance of responding to concerns, saying, "once we find a security issue, everything stops to get it fixed."

It is important to consider the possibility that participants were exaggerating their security emphasis in order to present a professional image during the interviews. We acknowledge this possibility, yet we maintain that the presence of established security teams and policies around external libraries serve as concrete indicators of an investment in security.

Examining particular security practices, most organizations we interviewed use standard encryption in their work. It is easy to make mistakes with custom do-it-yourself encryption, and many vulnerabilities identified by Reaves et al. [28] can be mitigated by using standard encryption algorithms and libraries. It is promising to see a majority of organizations adopting standard practices. Many of the other issues uncovered by Reaves et al. involve insecure connections to servers and are generally caused by improper use of SSL/TLS. The findings presented in the following section provide one possible explanation—that these issues may not be the fault of developers but rather due to strict partner specifications or governmental regulations.

5.3.2 Possible Causes of Vulnerabilities

During our conversations with the developers, we identified several possible causes for the lingering security vulnerabilities found in the current app ecosystem. Those possible problems are (1) the lack of a complete threat model, (2) the reliance on external or separate security teams, (3) budget constraints, (4) the frequent use of Stack Overflow, (5) limited security education, and (6) stringent requirements from partner organizations. Some of these "problems" may exhibit both pros and cons toward application security, so they should not be considered entirely harmful. For example, external security teams may inadvertently diffuse responsibility away from the actual developers, or if integrated well, they may provide important security benefits and expertise.

Incomplete threat model. From the academic mindset, a large focus of the security community centers around protecting the user. For DFS applications, this involves both protecting the customer's money from theft and protecting the customer's private data. Many of the interviewees emphasized primarily the need to protect the organization itself from theft. This is clearly also necessary, but considering only organizational security and ignoring the customer's interests would lead to many of the existing vulnerabilities that we and others have identified. When asked about the biggest security risk, for example, P1 responded:

> P1: The biggest security risk for the telco is leakage of revenue. The telco sells data bundles and airtime, so the major issue with security is when people have access to URLs that they can use to browse freely. For example, within my application there is a URL that is zero-rated, meaning it can be used even if you don't have airtime or a data bundle. If someone gets hold of that URL, they might be able to start browsing without paying, so that is a major risk.

When asked about specific protections for customers, P1 admitted that they sent a survey to users, and there "has been no concern about security." In fact, users expressed discontent over having to constantly log in to their accounts. In response, P1's company now links a user's SIM card and device to their account, so the users no longer need to log in. Asked about the possibility of a customer losing their phone or having it stolen, P1 said:

> P1: That concern doesn't arise at all because once your phone is stolen, you can go through a process of blocking and getting your SIM back... You can simply walk into our experience centers and get back on board. We will retrieve your number back in a very simple process.

Similarly, P3 agreed that the largest concerns are attacks on the company over the network. P3 was most worried about DDoS attacks because "when that happens we will be offline. For a financial institution, that is bad for business." When asked about the biggest security risks, P3 said:

> P3: There are two security risks and both are human. One is an ex-employee, and second is the customer. We consider a breach of a customer's account as a security issue, and we also consider remote access to unauthorized system data as a security risk, both to the customer and the employee. For myself, I have almost unlimited access to all mobile application development systems... I take part in API discussions and system architecture for a new system... With my experience and training, I would say if I left the company today, there are 900 security breaches, and I would be able to breach one of them.

P7, who displayed the most limited knowledge of security, admitted that the only sensitive info handled by the bank is the customer's "username and password." P7 admitted

	P1	P2	P3	P4	P5	P6/7
Dev. Experience (yrs)	7	8	8	12	1	2
Dev. Education	BS in Comp. Eng.	BS in Chem. Eng.	BS in Geo. Eng.	BS in CS	BS in CS	Engineer
Top 2 coding resources	SO, GitHub	SO, org. docs	SO, GitHub	SO, official docs	SO, coworkers	blogs, books
Threat model focus	network	network (MITM)	network (DDoS)	user fraud	network, user	user fraud
Org. Type	telco	bank	bank, software	software	telco	bank
Partnerships	none	unknown	banks	banks	none	telco
Org. Technical Training?	yes	yes	yes	yes	no	yes
Org. Code Review	outsourced	internal team	internal team	internal team	SonarQube	internal team
Org. Security Experts	outsourced	internal team	internal team	developer	internal team	internal team
Org. Security Training?	offered by contractor	offered by contractor	cert. from Kenya Bankers Assoc.	none	unknown	mandatory internal training
External Libraries?	yes	yes, with org. approval	yes, but dev. won't use them	yes, with org. approval	yes	no, against org. policy
Encryption Methods	custom	standard	standard	standard	standard	standard

Table 3: Summary of high-level results from interview participants. In the third row, SO refers to Stack Overflow. SonarQube is an open platform for code quality management available from `sonarqube.org`.

not knowing the most important security risks because "that part is another area, separate from mine."

Others, however, focused on attacks targeting users rather than attacks on the company network. P4 in particular held the stance that human fraud is the most volatile factor:

> P4: In most of the markets we're in, there isn't going to be a DDoS or packet sniffing, or any of the advanced stuff you see in the West. Here it is mostly users being manipulated into divulging key information. I don't lose sleep over somebody breaking into a system. It's more someone is going to give away this, or give away that, or disable something. It's just different in emerging markets... the weakest point is always the human factor... It hasn't happened to us, but in other banks the IT manager has just gone rogue and given up access to all the PINs. Here, it's not technical. The weakest point is always people.

External Security Teams. Every interviewee, except P4, noted that their organization has a dedicated security team separate from the primary development team. This is not inherently problematic, but as with P7 who did not have any model of security risks, relegating responsibility away from the developers may limit security features implemented in applications. Most security teams were distinct groups within the same organization, but in one case (P1), the telco uses an external contractor from China for both quality assurance and security specifications.

Budget Constraints. We found that organizations struggled with a trade-off between security and budget. With limited time or money, it may be more pertinent to implement a feature-rich application than a secure application, and this is especially true in emerging markets where companies' initial capital may be limited and new competitors move forward rapidly in the market. In particular, P6 said:

> P6: The main risk is that the budget is not enough. You need enough money for the project and to pay the developers. If the original project plan did not include all necessary features, then you have to pay more to the developers. Then you have to compromise to stay on schedule and make the launch date.

Stack Overflow.[1] We asked the developers to name the two resources they find most valuable while coding, and we gave as examples "books, coworkers, official documentation (such as Android docs), online forums (such as Stack Overflow), blogs, or other." All but one developer chose Stack Overflow, and they shared the consensus of using Stack Overflow in "almost all projects" (P1). Prior research has shown a correlation between the use of Stack Overflow and producing less secure code [8].

Education. As shown in Table 3, our interviewees had anywhere from 1–12 years of software development experience, and although all had a university degree, most participants stated that most of their programming skills were self-taught. For example, P2 admitted that schools "will only teach you the basics. Programming is about what you learn every day to get a job done. As a programmer, you just have to keep learning." P4 expressed similar sentiments about their own college experience:

> P4: The college experience wasn't really useful or helpful. When I got into actual employment, there was no moment where I said, 'ah thank god I learned that in college.'

Despite agreement among the interviewees about inadequate preparation for industrial software engineering, the prevalence of college degrees seems to indicate that the developers are well-educated and willing to learn the requisite skills to be successful. Accordingly, all but one participant, P5, said that their organization offers structured technical training for both new and veteran developers. Requirements for security experts, on the other hand, were much less clear and consistent, and only three participants (P3, P6, and P7) were aware of mandatory security training or requirements for security experts at their organizations.

Partner Requirements and Regulations. Both good and bad, the blame for certain vulnerabilities may not al-

[1]http://stackoverflow.com/

ways lie with the developer; rather, local government regulations or partner specifications may dictate certain insecure policies. During our interviews, all developers mentioned that application security decisions are made during the initial design phase, before implementation, and many times the developers are just matching specifications written by security teams or external partners. When asked about the importance of integrating security practices, P4 responded:

> P4: For us it's slightly complicated because a lot of the time we are at the mercy of our partner. Since we partner with a lot of banks, we will be at the mercy of their security team. We have our own ideas on how to secure our applications, but the level of security we have control over is only at the device level. Once an application communicates with the outside world, they can tell us, for example, you have to use a VPN or it must connect to these specific servers. . . for better or for worse a lot of the time our security review is us submitting required documentation to the bank audit team and meeting their spec.

Elaborating further, P4 claimed:

> P4: One of our strengths is biometry, but some banking partners will decide that they don't want to invest in the additional hardware required to do biometry. They are are fine with their single PINs.

In these scenarios, it is critical to examine the entire ecosystem to understand underlying issues, as competent developers may be forced to compromise their systems. This portrays developers in a better light, but the bad news is that even the best software teams and the best companies may be powerless to the prohibitive red tape, as P4 explained:

> P4: You will find a market where a gang of criminals exposed a particular human hack and made off with 2 million dollars. Then there will be this uproar and the government will just make a piece of regulation in reaction to that. So you will find this weird piece of regulation in this market that requires customers to go to the customer service location and present 7 forms of ID and their mother's DNA. . . That's emerging markets for you. . . We did one crazy one in West Africa where they didn't use any [encryption]. There again we are just at the mercy of the partner. . . We made them sign documents seven ways to Sunday because we were absolutely worried about [security]. What you'll find in these markets is that you have an IT person, and you are forced to work to their level of expertise.

5.3.3 Recommendations

Given these case-by-case insights, we propose several concrete and feasible steps forward. Our main suggestions revolve around external libraries, consistent security qualifications, and better resources to supplement Stack Overflow.

External Libraries. A tempting solution to rampant security vulnerabilities is to propose better external libraries, which will abstract away implementation details and integrate seamless encryption. Though these libraries are irrefutably useful, there are two problems with this catch-all solution. First is that many security concerns are human-related and depend heavily on context, such as proper login authentication or password reset procedures. Second, even though 5 out of our 6 developers were officially allowed to use external libraries, most developers were skeptical.

Although P3's organization does not explicitly forbid the use of external libraries, P3 said:

> P3: I avoid [external libraries] unless necessary, and I have never found a case where anything is necessary. In iOS, Android, Windows, and Blackberry, the native SDKs come with almost anything you need. I don't believe there is anything that's perfect and free. If you include someone's library and it is very good, you have to ask yourself why is he giving it out for free?

P2 and P4 both work in organizations which require any new external libraries to be approved. This adds a layer of consistency, but it places restrictions on the developers.

Security Qualifications. Each organization we studied had clear security experts, but for the most part, the developers had no knowledge of the actual qualifications of these experts. P6's organization holds mandatory training sessions for the security experts, and most interestingly, P3's organization requires the security team to have a national certification from the Kenya Bankers Association. Such a certification is not available in every country, but having a national or regional structure to provide security training and certifications could help standardize the ecosystem.

Supplemental Resources. We learned that developers are largely thinking about security issues. P6 in particular repeatedly asked the interviewers for suggestions on best practices to reset a user passwords. This poses an important question: when developers or product designers want to learn more about standard options for security measures, what resources should they use? Computer security is an art of balancing risk, so no single solution can apply to every situation. A set of guidelines, however, for financial services in the developing world could be both well-received and beneficial. Stack Overflow provides little consistency or standardization for best practices. Either more careful curation of Stack Overflow content or dedicated resource materials could provide a better starting point for developers.

6. DISCUSSION

Mobile money is rapidly expanding and offers great potential for global financial inclusion. In this section, we step back to discuss related work, outline the exposed security challenges, and propose future suggestions.

Related Work. Mobile security has been the subject of a relatively large body of work in the past decade. Of particular relevance to our work, Fahl et al. [16] interview developers about SSL/TLS mistakes and find that many issues stem from Android platform limitations rather than developer negligence. Other research has generated strategies to help Android developers [20], and several studies have measured the prevalence and danger of SSL mistakes in apps [15, 22].

On security in the developing world, Ben-David et al. [9] encourage researchers to devote more focus to the unique factors in the developing world. There has since been work in the areas of fraud and user authentication [25, 30], mobile applications [28], and end-to-end security [24, 26].

Security Challenges. Looking in-depth at our app analysis, we believe the most alarming threats include SMS spoofing and external libraries, which track users within a context where privacy perceptions are not well understood. Of limited yet realistic concern are SMS interceptions on the network, server attacks, MITM attacks, and unauthorized access from stolen devices. Attacks from malicious external apps and fake accounts are not high-priority concerns.

In our interviews we found that vulnerabilities may arise due to a combination of unique regional factors. The servers and specifications may be managed by multiple stakeholders with different requirements, resulting in restricted coverage of the threat model. Some developers also had limited threat models and only considered protecting the organization from fraud rather than protecting the customer. Developers who perceive human fraud as the greatest risk may not prioritize vulnerabilities which require technical expertise to exploit. We also believe that some errors stem from inadequate resources, such as Stack Overflow, which was the most widely-used resource and has been shown to foster insecure code.

Future Work. Reaves et al. [28] identified serious security flaws in a suite of mobile money apps, but many of these issues—SSL/TLS misconfigurations, information leakage on outdated Android versions, by known best practices. Competent organizational structures are in place, so additional reference sources in the form of guidelines for DFS products in the developing world or conscientious curation of Stack Overflow forums could be quickly integrated into developer workflows. Security certifications at a national level would also provide a useful validation for dedicated security teams within organizations.

We have uncovered some future challenges that may prove difficult to navigate. Institutional factors, such as disagreement among partner organizations or government and bank regulations, are hurdles that require more than better app development. One achievable objective is to pursue further research on a deployment-specific level of detail, such as password reset procedures, account-less money transfers, ID requirements, and other context-dependent issues.

7. CONCLUSION

We present a systematic threat model for the challenges associated with mobile money deployments in the developing world, and we use this threat model to guide a security analysis of 197 Android apps and 7 semi-structured interviews with developers. We argue that the threats addressed in prior work are not necessarily the most dire; rather, human-oriented fraud and restrictions from partner organizations present greater, more realistic challenges.

Acknowledgments

The University of Washington DFS Research Group is supported by a grant from the Bill & Melinda Gates Foundation.

8. REFERENCES

[1] Android Developers Dashboards. URL: https://developer.android.com/about/dashboards/index.html.

[2] Apktool: A tool for reverse engineering Android apk files. URL: http://ibotpeaches.github.io/Apktool/.

[3] StatCounter Global Stats. URL: http://gs.statcounter.com.

[4] World Bank. 2016. World Development Report 2016: Digital Dividends. Washington, DC: World Bank.

[5] Telco 2.0, Security Breach at M-PESA: Telco 2.0 Crash Investigation, Feb. 2010. URL: http://www.telco2.net/blog/2010/02/security_breach_at_mpesa_telco.html.

[6] Kenya to switch off 'fake' mobile phones. *BBC News*, Sept. 2012. URL: www.bbc.com/news/technology-19731514.

[7] Safaricom Official Blog, Dealing with fraudsters, May 2015. URL: www.safaricom.co.ke/blog/dealing-with-fraudsters/.

[8] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky. You Get Where You're Looking For: The Impact Of Information Sources on Code Security. In *IEEE Symposium on Security and Privacy*, 2016.

[9] Y. Ben-David, S. Hasan, J. Pal, M. Vallentin, S. Panjwani, P. Gutheim, J. Chen, and E. A. Brewer. Computing Security in the Developing World: A Case for Multidisciplinary Research. NSDR, 2011.

[10] J. Chen, M. Paik, and K. McCabe. Exploring Internet Security Perceptions and Practices in Urban Ghana. SOUPS, 2014.

[11] K. Chen, X. Wang, Y. Chen, P. Wang, Y. Lee, X. Wang, B. Ma, A. Wang, Y. Zhang, and W. Zou. Following Devil's Footprints: Cross-Platform Analysis of Potentially Harmful Libraries on Android and iOS. In *IEEE Symposium on Security and Privacy*, 2016.

[12] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner. Analyzing Inter-application Communication in Android. MobiSys, 2011.

[13] C. Cobb, S. Sudar, N. Reiter, R. Anderson, F. Roesner, and T. Kohno. Computer Security for Data Collection Technologies. ICTD, 2016.

[14] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones. OSDI, 2010.

[15] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith. Why Eve and Mallory Love Android: An Analysis of Android SSL (in)Security. CCS, 2012.

[16] S. Fahl, M. Harbach, H. Perl, M. Koetter, and M. Smith. Rethinking SSL Development in an Appified World. CCS, 2013.

[17] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android Permissions Demystified. CCS, 2011.

[18] GSMA. Mobile Money Deployment Tracker. URL: http://www.gsma.com/mobilefordevelopment/m4d-tracker/mobile-money-deployment-tracker.

[19] Joseck Luminzu Mudiri. Fraud in Mobile Financial Services. Technical report, MicroSave, Dec. 2012.

[20] M. Linares-Vásquez. Supporting Evolution and Maintenance of Android Apps. ICSE, 2014.

[21] K. McKee, M. Kaffenberger, and J. M. Zimmerman. Doing Digital Finance Right: The Case for Stronger Mitigation of Customer Risks. Technical Report 103, June 2015.

[22] L. Onwuzurike and E. De Cristofaro. Danger is My Middle Name: Experimenting with SSL Vulnerabilities in Android Apps. WiSec, 2015.

[23] M. Paik. Stragglers of the Herd Get Eaten: Security Concerns for GSM Mobile Banking Applications. HotMobile, 2010.

[24] S. Panjwani. Towards End-to-end Security in Branchless Banking. HotMobile, 2011.

[25] S. Panjwani and E. Cutrell. Usably Secure, Low-cost Authentication for Mobile Banking. SOUPS, 2010.

[26] S. Panjwani, M. Ghosh, P. Kumaraguru, and S. V. Singh. The Paper Slip Should Be There!: Perceptions of Transaction Receipts in Branchless Banking. MobileHCI, 2013.

[27] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner. AdDroid: Privilege Separation for Applications and Advertisers in Android. ASIACCS, 2012.

[28] B. Reaves, N. Scaife, A. Bates, P. Traynor, and K. R. B. Butler. Mo(bile) Money, Mo(bile) Problems: Analysis of Branchless Banking Applications in the Developing World. In *USENIX Security*, 2015.

[29] B. Reaves, N. Scaife, D. Tian, L. Blue, P. Traynor, and K. R. B. Butler. Sending out an SMS: Characterizing the Security of the SMS Ecosystem with Public Gateways. In *IEEE Symposium on Security and Privacy*, 2016.

[30] A. Sharma, L. Subramanian, and D. Shasha. Secure Branchless Banking. NSDR, 2009.